Разработка и анализ алгоритмов

Лекция 9
Хеш-таблицы

Сергей Леонидович Бабичев

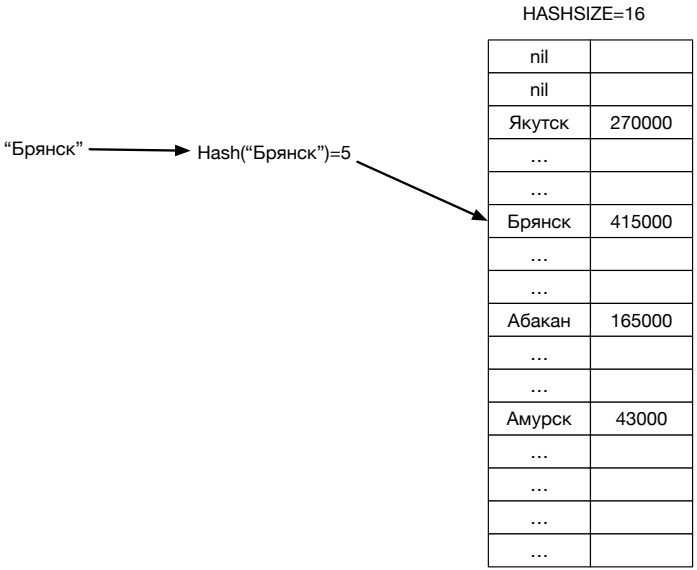
Хеш-таблицы

Хеш-таблицы

- Задача разбиения пространства ключей (*partitioning*) решается хеш-функциями.
- Организация хранения данных имеет несколько вариантов.
- Прямолинейное разбиение на независимые вторичные поисковые структуры называется *закрытой адресацией*. Другие названия: *цепочки переполнения*, *хеширование с цепочками*.
- Имеется таблица с прямым доступом (указатели) для доступа к вторичной поисковой структуре в которой и производится основной поиск.
- Именно во вторичных поисковых структурах хранятся ключи и значения.
- Хранение ключей и значений внутри самой таблицы — *открытая адресация*. Другое название — *опробывание*.

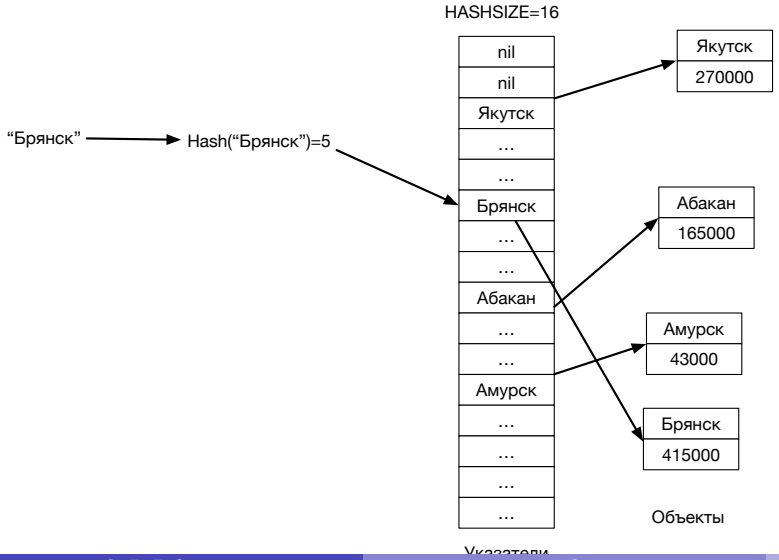
Закрытая адресация

- Простая хеш-таблица с закрытой адресацией.



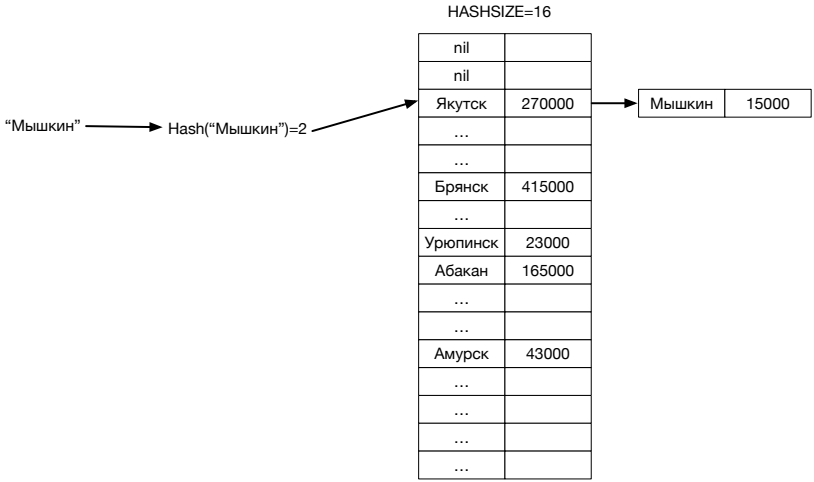
Закрытая адресация

- Хеш-таблица с закрытой адресацией, обычная реализация в виде массива указателей.



Закрытая адресация

- При коллизии во время создания элемента можно использовать любую поисковую структуру данных.
- Пусть это будет связный список конфликтующих.



Закрытая адресация: операция *insert*

1. Вычисляется хеш-функция от ключа.
2. Определяется слот возможного хранения — вторичная поисковая структура данных.
3. Если вторичной структуры нет, то она создаётся.
4. Элемент добавляется во вторичную структуру.

Закрытая адресация: операция *find*

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

Закрытая адресация: операция *delete*

1. Вычисляется хеш-функция ключа.
2. Определяется вторичная поисковая структуре данных.
3. Если вторичной структуры нет, то нет и элемента.
4. Иначе элемент удаляется из вторичной структуры.
5. Если вторичная структура теперь пуста, удаляется точка входа.

Закрытая адресация: сложность

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

Lemma

Матожидание числа операций сравнения ключа при закрытой адресации со связными списками оценивается в $1 + \frac{\alpha}{2}$.

Доказательство.

Одно сравнение происходит, если ключ находится в начале списка. Матожидание ровно одного сравнения есть $\frac{C}{M} = \frac{1}{\alpha}$. Второй элемент списка имеется только при совпадении значение $H(key_1) = H(key_2)$, что примем за $\frac{1}{M}$. Тогда

$$\frac{1}{C}\sum_{i=1}^{C}\left(1 + \sum_{j=i+1}^{C}\frac{1}{M}\right) = 1 + \frac{1}{C}\sum_{i=1}^{C}\sum_{j=i+1}^{C}\frac{1}{M} =$$

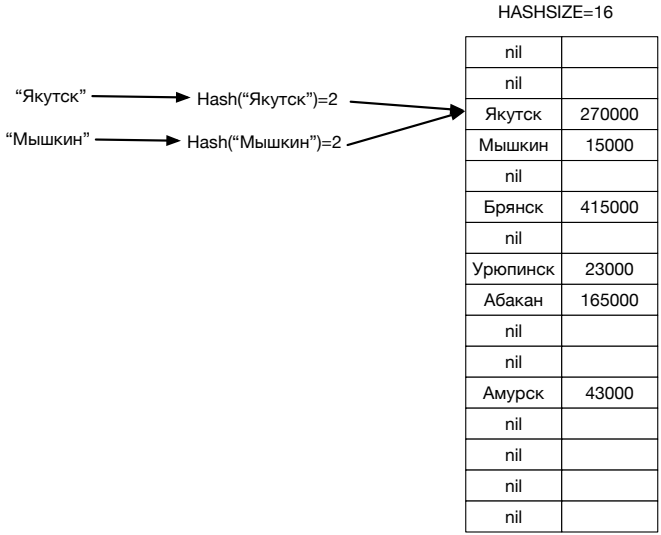
$$= 1 + \frac{1}{CM}\sum_{i=1}^{C}(C-i) = 1 + \frac{1}{CM}\left(C^2 - \frac{C(C+1)}{2}\right) =$$

$$= 1 + \frac{C-1}{2M} = 1 + \frac{\alpha}{2} - \frac{\alpha}{2C}$$

□

Хеш-таблицы с открытой адресацией

- Другой способ поиска — хранить всё в одной таблице повторно и искать в этой же таблице где-то в другом месте.



Хеш-таблицы с открытой адресацией

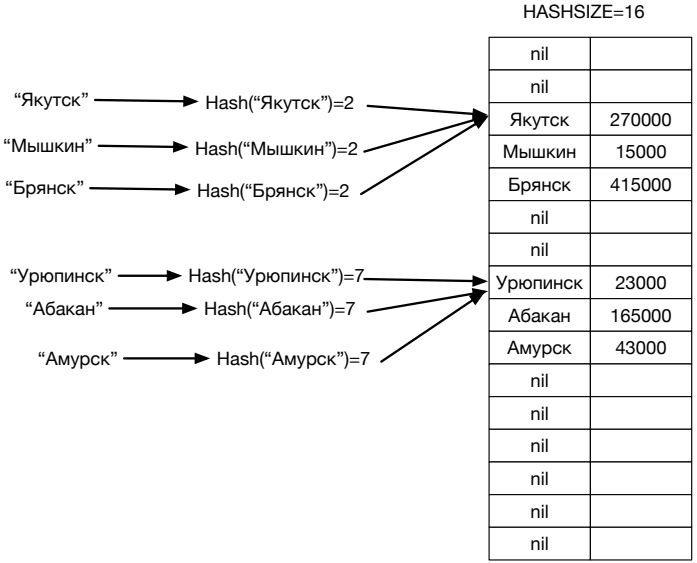
1. При поиске существующего вычисляется хеш-функция.
2. Определяется место поиска — индекс в хеш-таблице.
3. Если по индексу ничего нет, то нет и элемента.
4. Иначе по индексу — элемент с нашим ключом — элемент найден.
5. Если по индексу — элемент с другим ключом или элемент помечен удалённым, индекс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

Открытая адресация: сложность неудачного поиска

- При последовательном опробовании среднее время поиска зависит от организации занятых ячеек в непрерывные области — кластеры.
- Пусть таблица занята наполовину.
- Случай 1. Чётные заняты, нечётные свободны. Средняя длина кластера $\frac{1}{2}$, среднее количество поисков $1 + \frac{1}{2}$.
- Случай 2. Первая половина занята, вторая свободна. Средняя длина кластера $\frac{1}{2}$, среднее количество поисков $1 + \frac{C + (C-1) + (C-2) + \ldots}{2C} \approx \frac{C}{4}$.

Открытая адресация: сложность неудачного поиска

Lemma (О матожидании количества попыток при неудаче)

Матожидание количества попыток при неудачном поиске в хеш-таблице с открытой адресацией и коэффициенте заполнения α не превышает $\dfrac{1}{1-\alpha}$.

Доказательство.

Каждый неудачный поиск завершается пустой ячейкой. Случайное событие A_i — i-я попытка пришлась на занятую ячейку. Тогда требуемая величина $X_i = A_1 \cap A_2 \dots A_i$.

$$p(X_i) = p(A_1 \cap A_2 \dots A_{i-1}) = p(A_1) \cdot p(A_2|A_1) \cdot p(A_3|(A_2 \cap A_1) \cdot \dots$$

$p(A_1) = \alpha$. Вероятность $j-$й попытки над заполненной ячейкой $\frac{C-j-1}{M-j-1}$. Поэтому

$$p(X_i) = \frac{C}{M} \cdot \frac{C-1}{M-1} \cdot \frac{C-i+2}{M-i+2} \leqslant \left(\frac{C}{M}\right)^{i-1} = \alpha^{i-1}$$

Суммируя $1 + \alpha + \alpha^2 + \dots$ получаем $\dfrac{1}{1-\alpha}$. □

Открытая адресация: сложность удачного поиска

Lemma (О матожидании количества попыток при успехе)

Матожидание количества попыток при удачном поиске в хеш-таблице с открытой адресацией и коэффициенте заполнения α не превышает $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$.

Доказательство.

После вставки i ключей матожидание количества проб при поиске не превышает $\frac{M}{M-i}$. Усредним по всем ключам:

$$\frac{1}{C}\sum_{i=0}^{C-1}\frac{M}{M-i} = \frac{1}{\alpha}\sum_{k=M-C+1}^{M}\frac{1}{k} \leqslant \frac{1}{\alpha}\int_{M-C}^{M}\frac{1}{x}dx = \frac{1}{\alpha}\ln\frac{M}{M-C} = \frac{1}{\alpha}\ln\frac{1}{1-\alpha}.$$

□

Несколько чисел

α	Закрытая	Открытая неуспех	Открытая успех
0.1	1.05	1.11	1.05
0.2	1.10	1.25	1.12
0.3	1.15	1.43	1.19
0.4	1.20	1.67	1.28
0.5	1.25	2.00	1.39
0.6	1.30	2.50	1.53
0.7	1.35	3.33	1.72
0.8	1.40	5.00	2.01
0.9	1.45	10.00	2.56

Расширение хеш-таблиц

Когда α начинает превосходить 0.5-0.6 таблицу расширяют.

- Создаётся пока пустая хеш-таблица с примерно удвоенным размером.
- Из оригинального массива в порядке увеличения индексов извлекаются элементы и вставляются в новую таблицу.
- Старая таблица 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.
- 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: cuckoo-хеширование

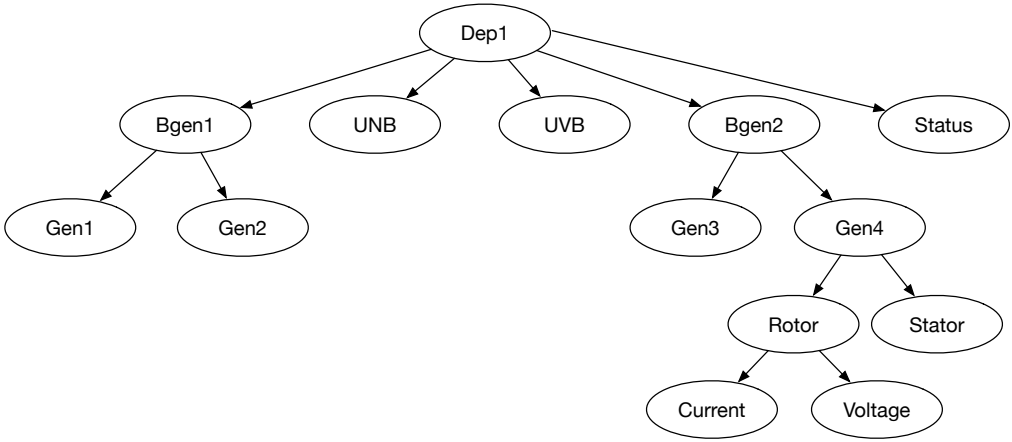
- Рассмотренный вариант организации хеш-таблиц всегда имеет ровно одно предпочтительное место для ключа.
- При cuckoo-хешировании для каждого ключа имеется две таблицы и два места, определяемых двумя хеш-функциями — $P_1 = H_1(key)$ и $P_2 = H_2(key)$.
- Если оказывается коллизия в позиции P_1, то производится 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.

Сочетание хеш-таблиц и деревьев

Хеш-таблицы и деревья

Задача:

- Имеется набор объектов, представляющих иерархию гидроэлектростанции (фрагмент)



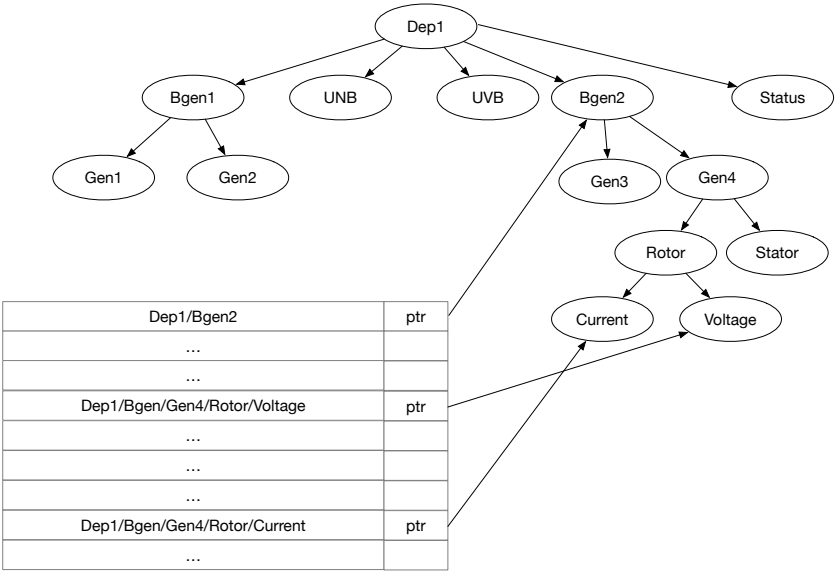
Dep1/Bgen/Gen4/Rotor/Voltage=127

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.



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

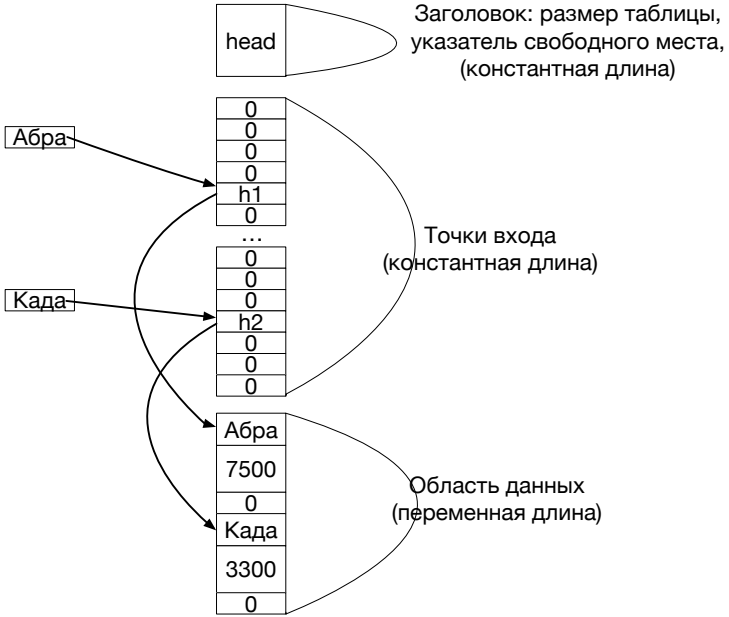
Данные располагаются в 5000000 файлах, в каждом из которых по 1000 строк.

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

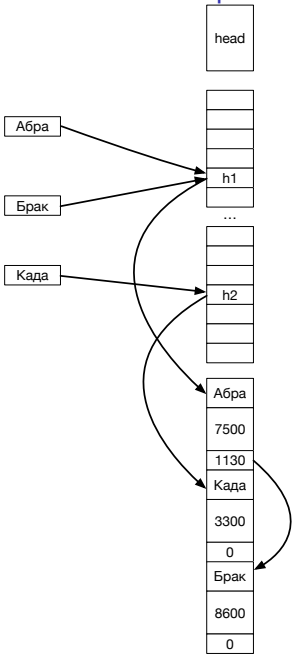
Хеш-таблицы во внешней памяти

- Общий размер превышает ТВ.
- 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.