Хеш-таблица

ЛЕКЦИЯ 10

Что есть хеш-таблица?

Хеш-таблицей называется структура данных, обеспечивающая очень быструю вставку и поиск. На первый взгляд звучит слишком хорошо, чтобы быть правдой: независимо от количества элементов данных вставка и поиск (а иногда и удаление) выполняются за время, близкое к постоянному: O(1) в O-синтаксисе. На практике это лишь несколько машинных команд.

Для пользователя хеш-таблицы обращение к данным происходит практически мгновенно. Все делается настолько быстро, что компьютерные программы часто используют хеш-таблицы при необходимости сделать выборку из десятков тысяч элементов менее чем за секунду (как, например, в системах проверки орфографии).

Что есть хеш-таблица?

Хеш-таблицы по скорости значительно превосходят деревья, которые, как вы узнали в предыдущих главах, выполняют операции за с относительно малое время O(logN). Операции с хеш-таблицами не только быстро выполняются, но и относительно просто программируются.

Недостатки хеш-таблиц

У хеш-таблиц также имеются свои недостатки. Они реализуются на базе массивов, а массивы трудно расширить после создания. У некоторых разновидностей хеш-таблиц быстродействие катастрофически падает при заполнении таблицы, поэтому программист должен довольно точно представлять, сколько элементов данных будет храниться в таблице

Кроме того, при работе с хеш-таблицами не существует удобного способа перебора элементов в определенном порядке (скажем, от меньших к большим). Если вам необходима такая возможность, поищите другую структуру данных

Хеширование

- Допустим, вы хотите сохранить в оперативной памяти словарь английского языка на 50 000 слов. Каждое слово должно занимать отдельную ячейку массива, чтобы к каждому слову можно было обратиться по его индексу. Доступ к данным будет очень быстрым, но как связать индексы со словами? Скажем, вы хотите найти в словаре слово morphosis; как определить его индекс?
- Нам нужна система, которая преобразует слово в соответствующий ему индекс. Как известно, компьютеры используют разные схемы числового представления отдельных символов.

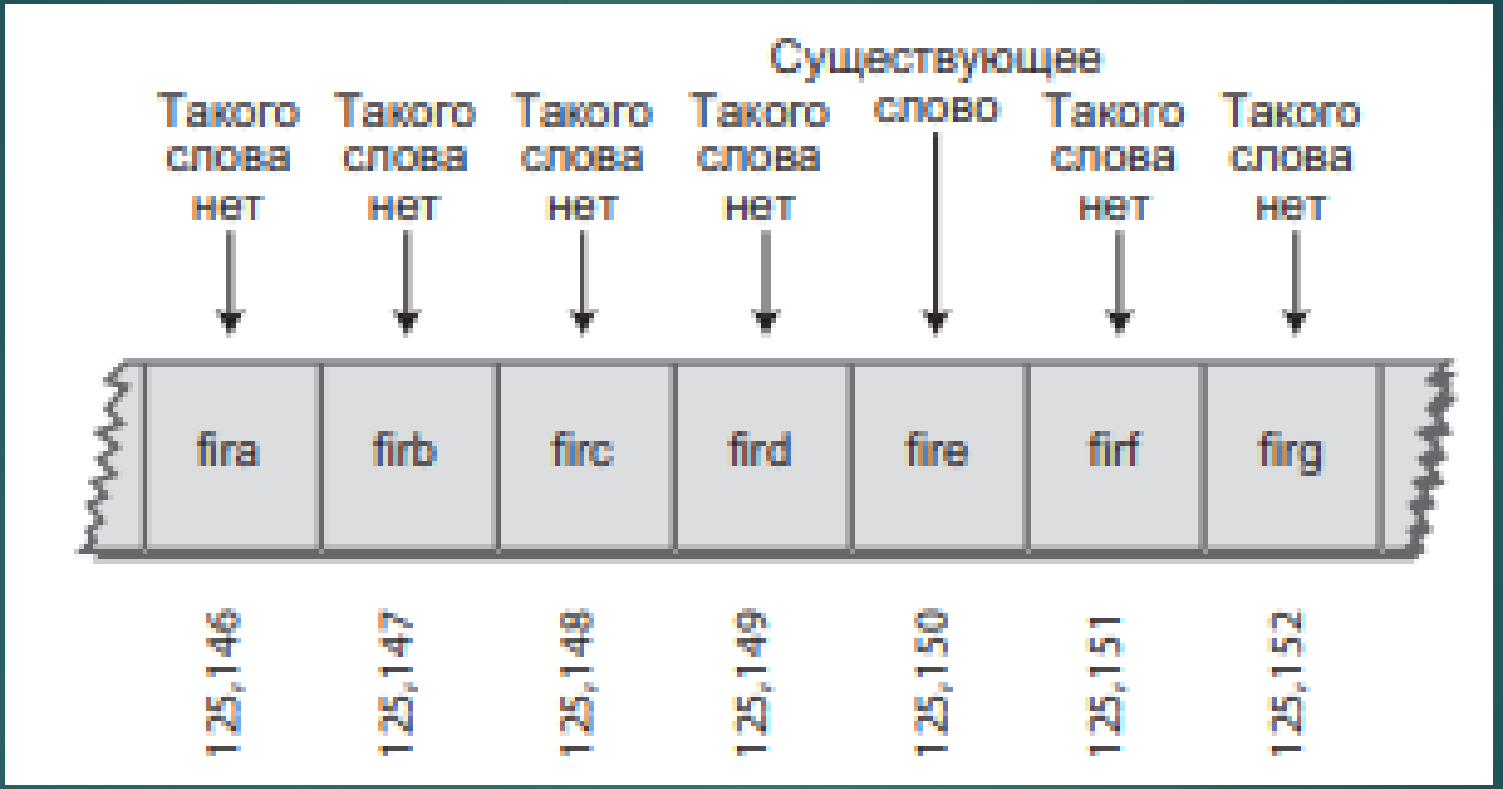
Хеширование

- Например, поступим со словом так: разложим его на буквы, преобразуем буквы в числовые эквиваленты, умножим их на соответствующие степени 27 (так как существует 27 возможных символов, включая пробелы) и сложим результаты. В результате для каждого слова генерируется уникальное число. Допустим, нам потребовалось преобразовать слово cats в число. Мы заменяем буквы цифрами так, как было показано ранее, после чего умножаем каждое число на соответствующую степень 27 и суммируем результаты: 3*273 + 1*272 + 20*271 + 19*270 . Вычисление степеней дает 3*19 683 + 1*729 + 20*27 + 19*1, а в результате суммирования кодов букв, умноженных на степени, мы получаем 59 049 + 729 + 540 + 19 = 60 337

Хеширование

- Эта схема действительно генерирует уникальное число для каждого потенциального слова. Мы только что вычислили числовой эквивалент слова из четырех букв. А что произойдет со словами большего размера? К сожалению, диапазон чисел становится очень большим. Для больших слов такие уникальные индексы превышают значения в несколько триллионов, поэтому нам нужно как-то сократить систему, учитывая несуществующие слова

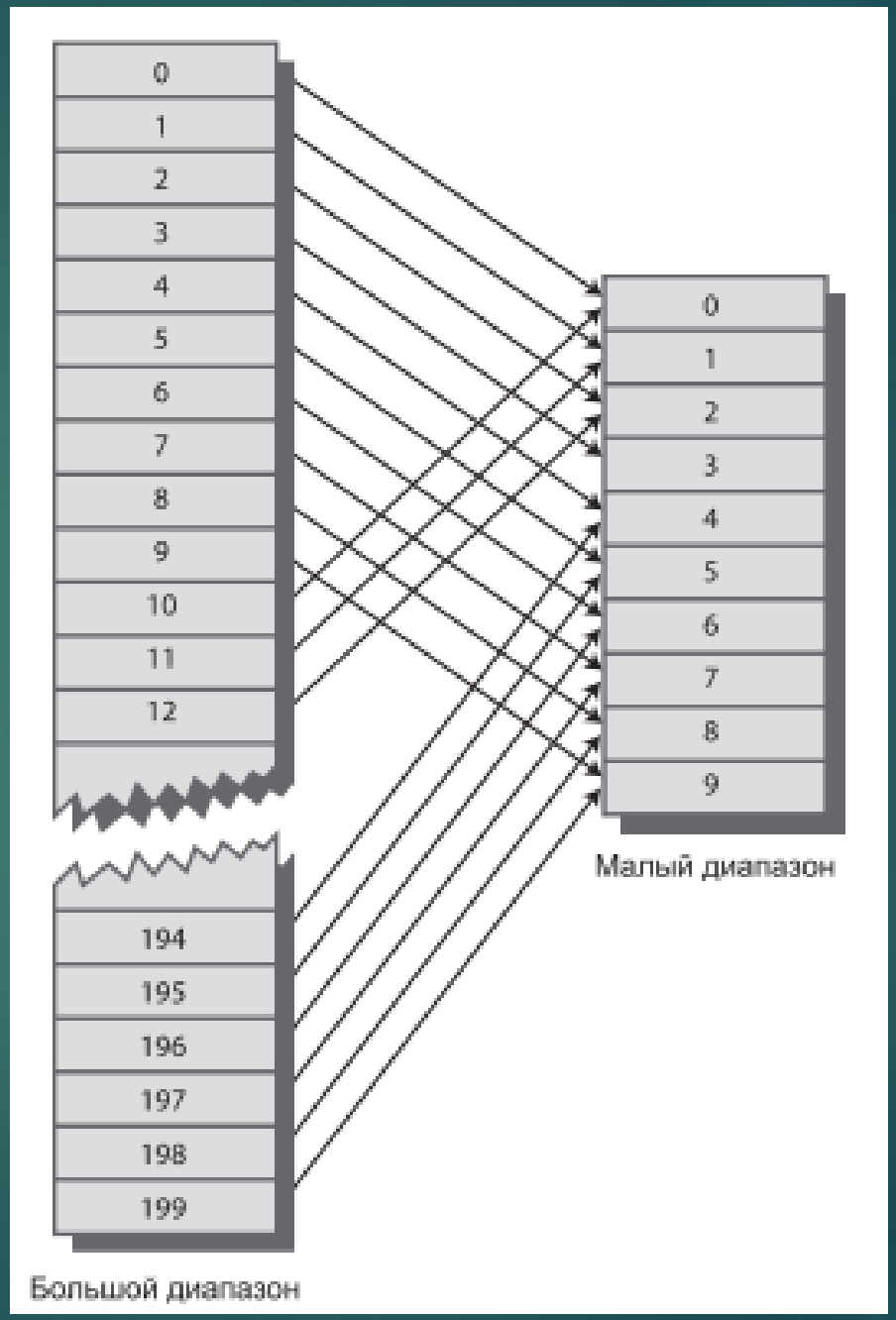
Хеширование



Хеширование

- Остатки от деления любого числа на 10 всегда лежат в диапазоне от 0 до 9; например, результат выражения 13%10 равен 3, а результат 157%10 равен 7 (рис. 11.3). Диапазон 0–199 преобразуется в диапазон 0–9, то есть происходит сжатие с коэффициентом 20:1. Аналогичное выражение может использоваться для сжатия очень больших чисел, однозначно представляющих каждое слово английского языка, в индексы массива: arrayIndex = hugeNumber % arraySize; Это выражение является примером хеш-функции — функции, преобразующей число из большего диапазона в число из меньшего диапазона. Меньший диапазон соответствует индексам массива. Массив, в который вставляются данные с использованием хеш-функции, называется хеш-таблицей.

Хеширование



Коллизии

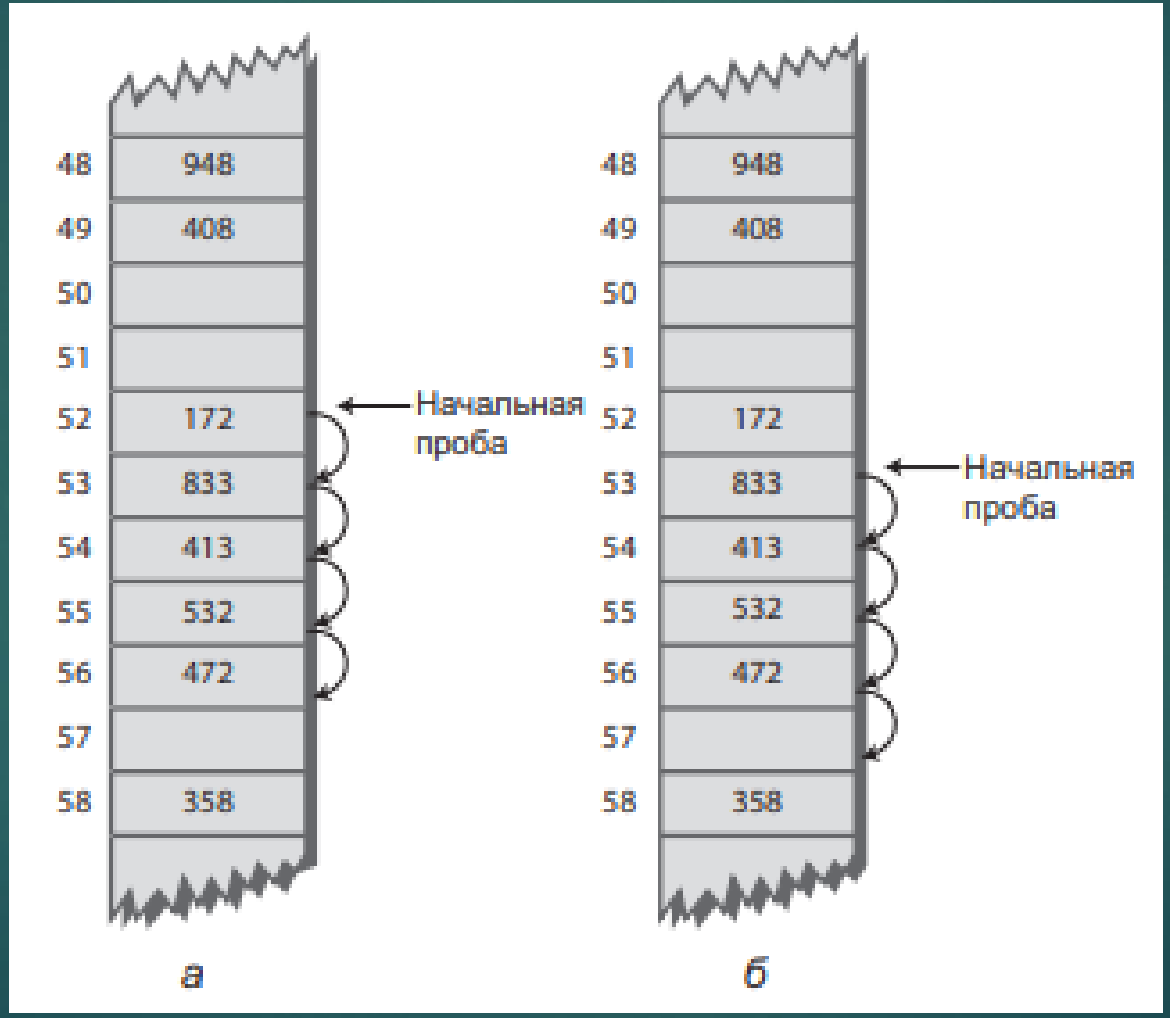
Сокращение диапазона значений имеет свою цену. Схема уже не гарантирует, что после сжатия два слова не будут хешироваться в один индекс массива. Нечто похожее происходило и при суммировании кодов букв, но тогда все было гораздо хуже. При суммировании было всего 260 возможных результатов (для слов длиной до 10 букв), а теперь диапазон расширяется до 50 000 возможных результатов. Впрочем, даже в этом случае невозможно избежать хеширования нескольких разных слов в один индекс массива (по крайней мере изредка). Мы надеялись, что на один индекс будет приходиться один элемент данных, но как оказалось, это невозможно. Лучшее, на что можно надеяться, что не слишком много слов будет хешироваться в один индекс.

Вспомните, что при определении массива количество зарезервированных ячеек вдвое превышало количество элементов данных. Таким образом, приблизительно половина ячеек остается пустой. Одно из возможных решений при возникновении коллизии заключается в систематизированном поиске пустой ячейки и вставке нового элемента в нее (вместо индекса, полученного в результате применения хеш-функции). Такое решение называется открытой адресацией.

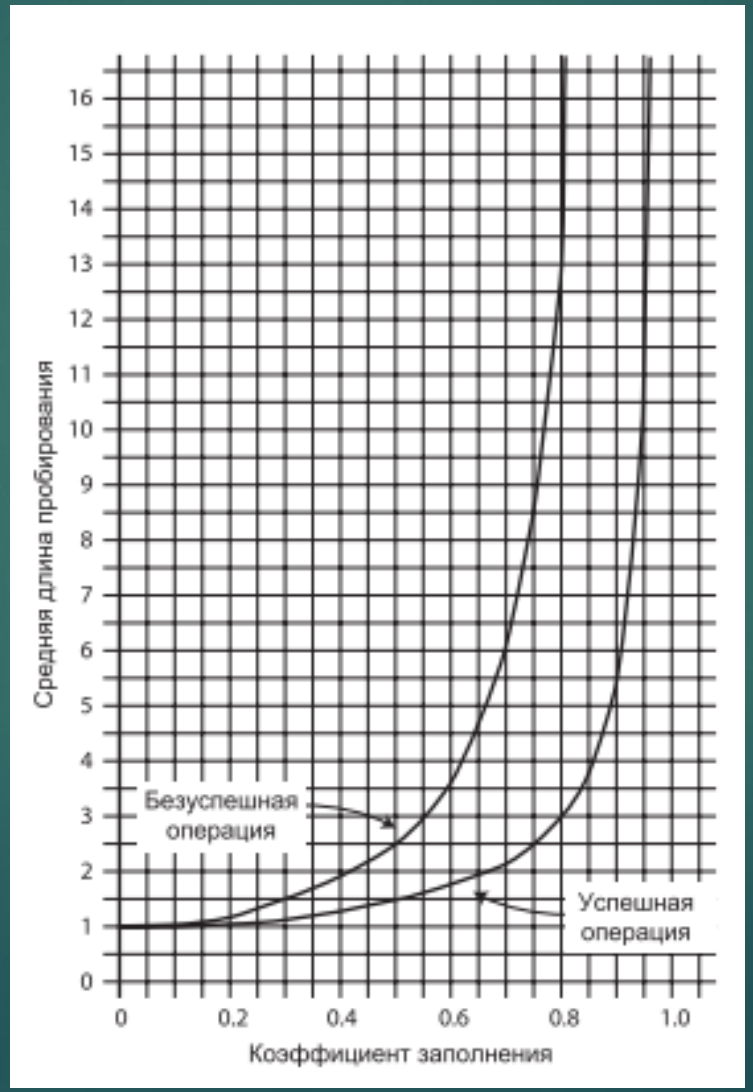
Коллизии

Во втором решении (упоминавшемся ранее) создается массив, содержащий связанные списки слов вместо самих слов. При возникновении коллизии новый элемент просто вставляется в список с соответствующим индексом. Этот метод называется методом цепочек. Мы рассмотрим открытую адресацию и метод цепочек, а затем вернемся к теме хеш-функций.

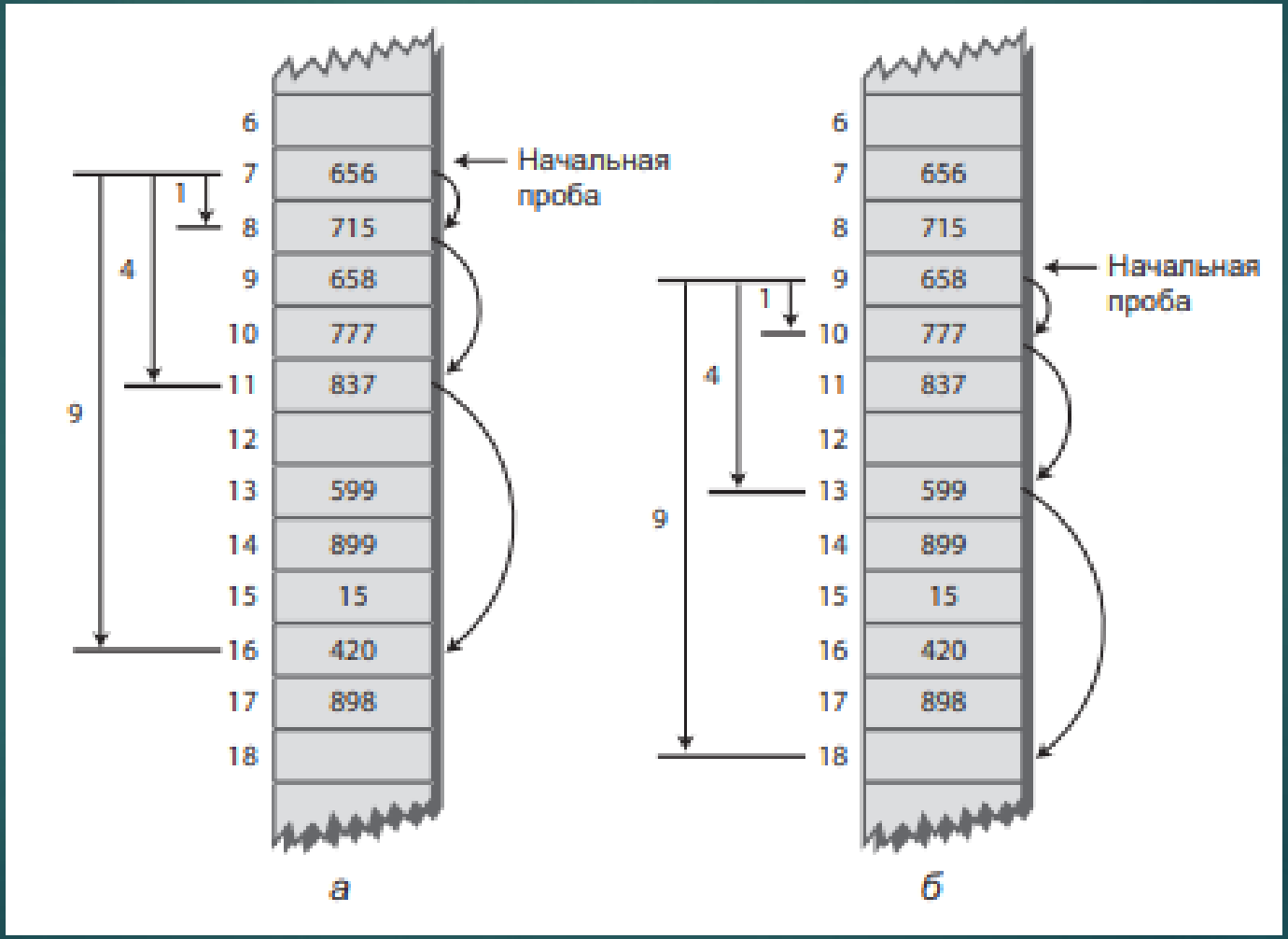
Открытая адресация, линейное пробирование



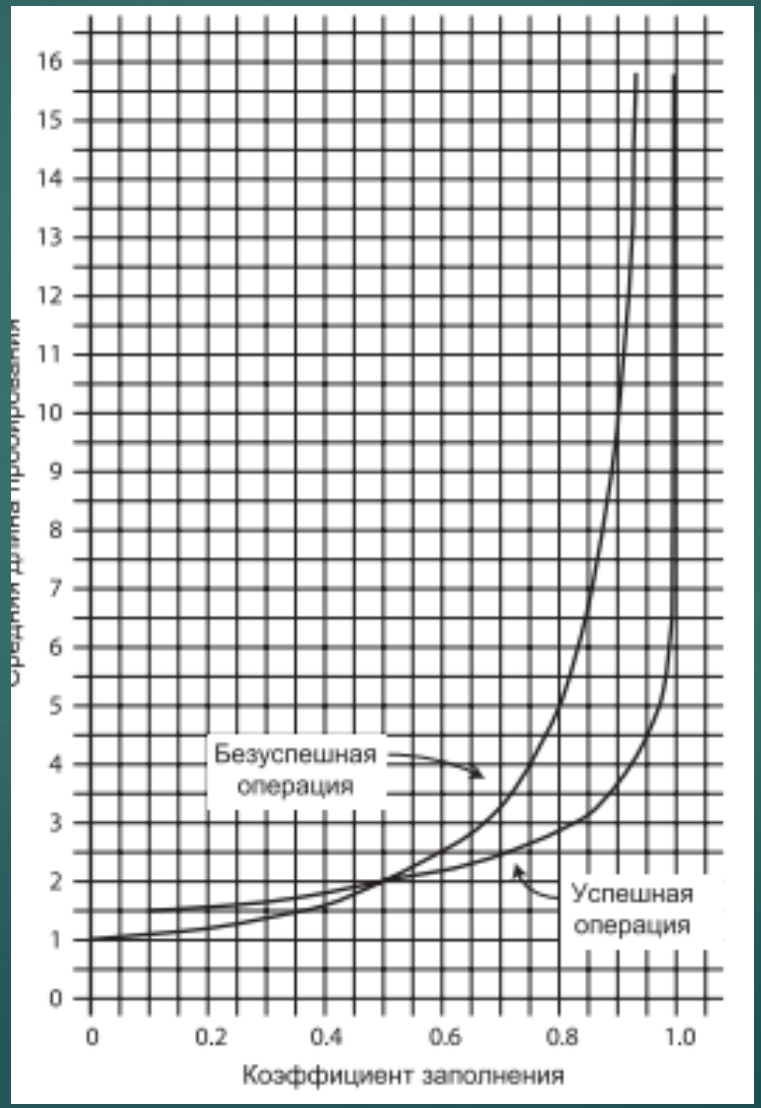
Открытая адресация, линейное пробирование - эффективность



Открытая адресация, квадратичное пробирование



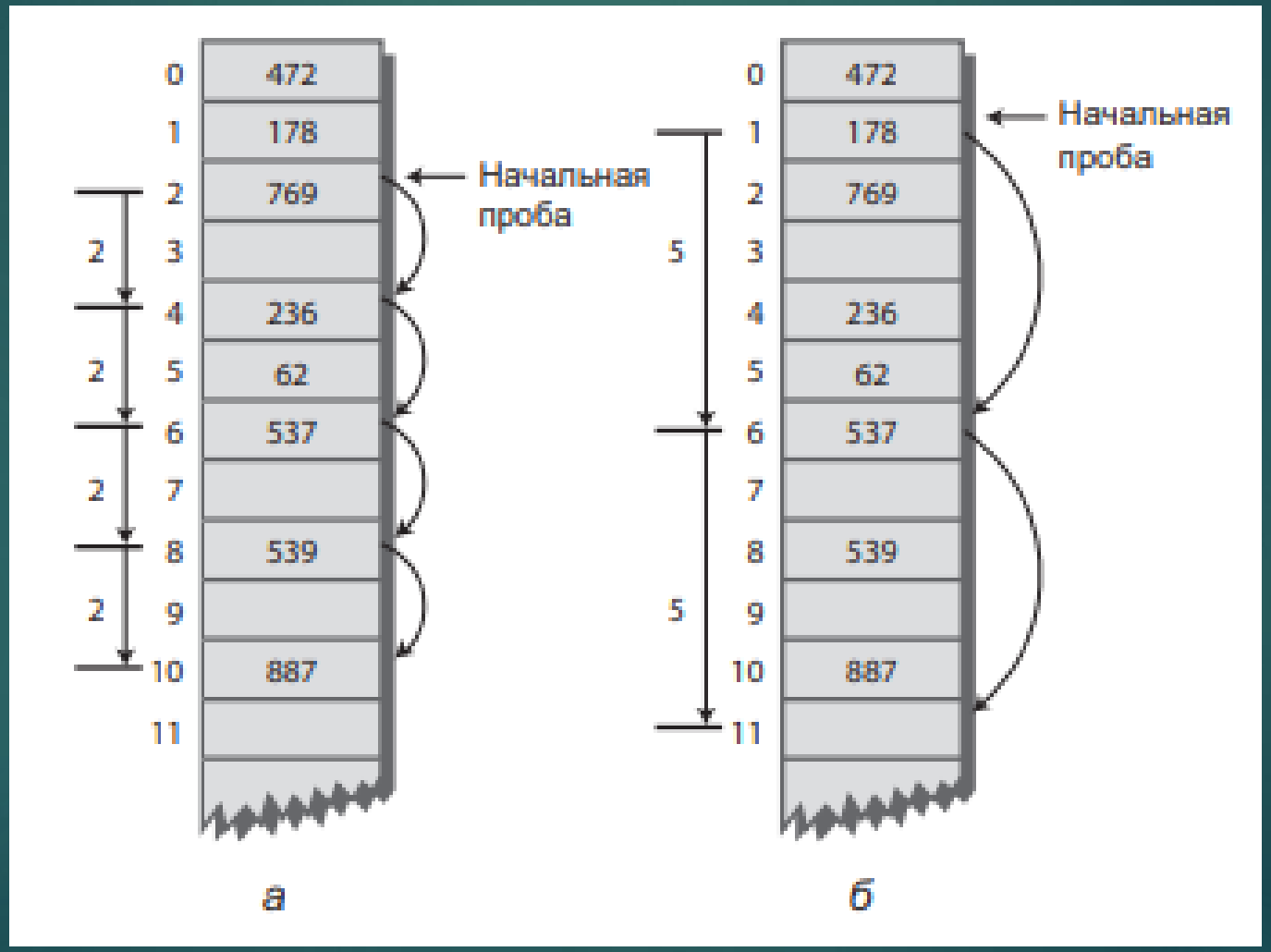
Открытая адресация, квадратичное пробирование - эффективность



Открытая адресация, квадратичное пробирование

При линейном пробировании, если первичный индекс хеширования равен x, то последующие пробы проверяют позиции x + 1, x + 2, x + 3 и т. д. При квадратичном пробировании проверяются позиции x + 1, x + 4, x + 9, x + 16, x + 25 и т. д. Расстояние от исходной позиции вычисляется как квадрат номера шага: x + 1*1 , x + 2*2 , x + 3*3 , x + 4*4; , x + 5*5 и т. д.

Открытая адресация, двойное хеширование



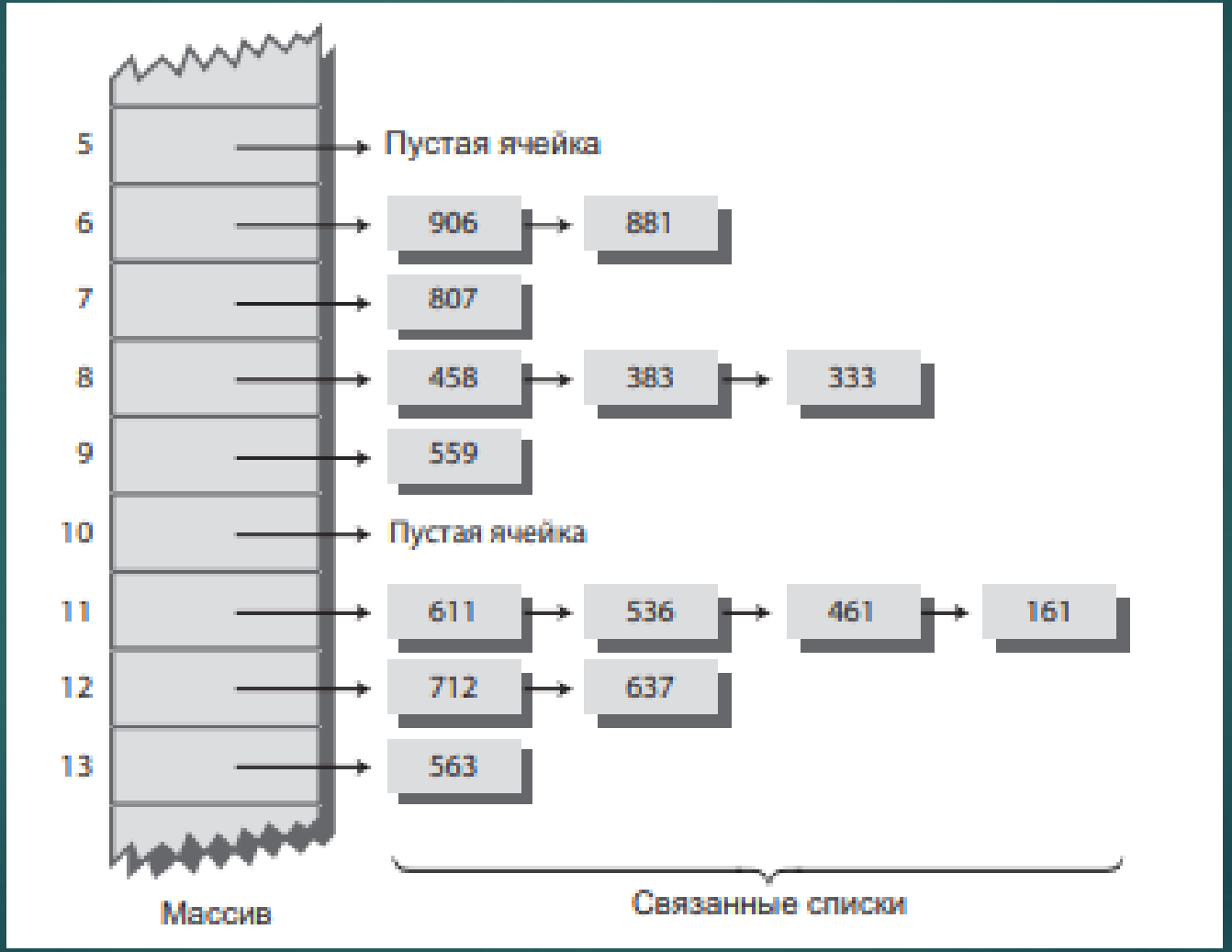
Двойное хеширование

При использовании двойного хеширования размер таблицы должен быть простым числом. Чтобы понять смысл этого требования, представьте ситуацию, в которой размер таблицы простым числом не является. Предположим, размер массива равен 15 (индексы от 0 до 14), а конкретный ключ хешируется в исходный индекс 0 со смещением 5. Пробы будут выполняться в последовательности 0, 5, 10, 0, 5, 10 и т. д. до бесконечности. Проверяются только эти три ячейки, поэтому алгоритм «не увидит» пустые ячейки 1, 2, 3 и т. д., а из попытки выполнения операции ничего не выйдет. Если бы размер массива был равен 13 (простое число), то в процессе пробирования в конечном итоге была бы проверена каждая ячейка: 0, 5, 10, 2, 7, 12, 4, 9, 1, 6, 11, 3 и т. д

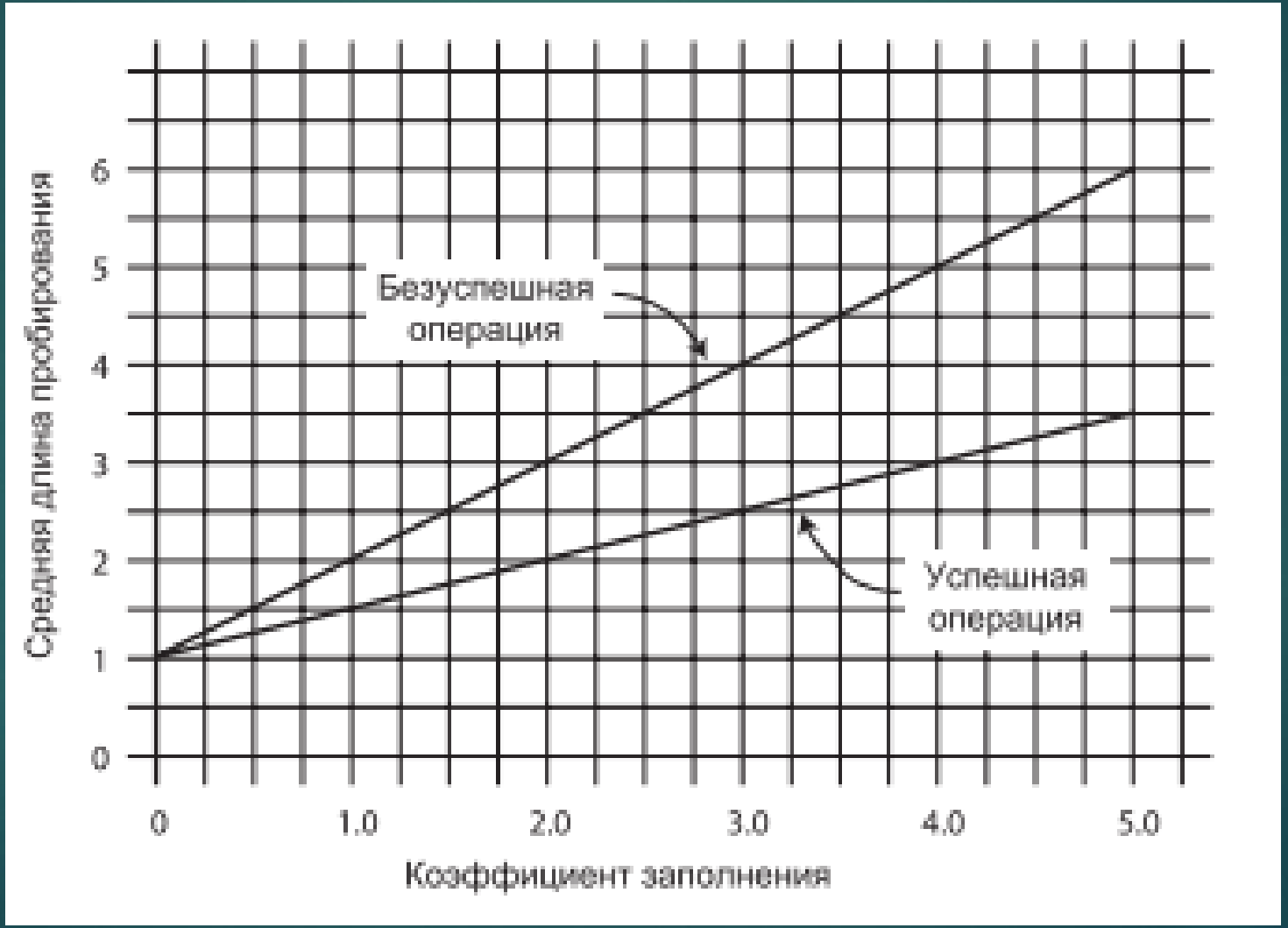
Двойное хеширование

. Даже если в массиве имеется всего одна пустая ячейка, она будет успешно обнаружена. Простой размер массива не делится нацело ни на какое число, поэтому последовательность проб рано или поздно проверит каждую ячейку. Аналогичный эффект проявляется с квадратичным пробированием. Однако в этом случае смещение увеличивается с каждым шагом, что в конечном итоге приведет к переполнению переменной, в которой оно хранится (хотя это тоже приведет к прерыванию бесконечного цикла). В общем случае при использовании открытой адресации предпочтение следует отдавать двойному хешированию.

Закрытая адресация, цепочки (списки взаимозаменяемы с массивами)



Закрытая адресация, метод цепочек - эффективность



Двойное хеширование

. Даже если в массиве имеется всего одна пустая ячейка, она будет успешно обнаружена. Простой размер массива не делится нацело ни на какое число, поэтому последовательность проб рано или поздно проверит каждую ячейку. Аналогичный эффект проявляется с квадратичным пробированием. Однако в этом случае смещение увеличивается с каждым шагом, что в конечном итоге приведет к переполнению переменной, в которой оно хранится (хотя это тоже приведет к прерыванию бесконечного цикла). В общем случае при использовании открытой адресации предпочтение следует отдавать двойному хешированию.

Выбор хеш-функции

Хорошая хеш-функция должна быть простой, чтобы ее результат быстро вычислялся. Главное преимущество хеш-таблиц — скорость. Если хеш-функция работает медленно, то это преимущество будет утрачено. Хеш-функция с многочисленными операциями умножения и деления вряд ли хорошо справится со своим делом.

Основной целью хеш-функции является преобразование диапазона ключей в диапазон индексов, обеспечивающее равномерное распределение ключей по индексам хеш-таблицы. Ключи могут быть как полностью случайными, так и частично детерминированными.

Выбор хеш-функции

Идеальная хеш-функция ставит в соответствие каждому значению ключа уникальный индекс. Такое поведение возможно только при нетипично «хорошем» поведении ключей с диапазоном, достаточно узким для прямого использования в качестве индексов (как в примере с табельными номерами работников в начале главы). На практике обычно ни одно из этих условий не выполняется, и хеш-функции приходится сжимать широкий диапазон ключей в меньший диапазон индексов. Желательное поведение хеш-функции зависит от распределения значений ключей в конкретной базе данных. Ранее мы предполагали, что данные равномерно распределены по всему диапазону. Для такой ситуации хеш-функция вида индекс = ключ % размер_массива; достаточно хорошо работает. В ней используется только одна математическая операция, поэтому для действительно случайных ключей полученные индексы тоже будут случайными, а следовательно, будут иметь хорошее распределение.

Выбор хеш-функции

Поля ключа необходимо по возможности «сжать», чтобы каждый их бит нес полезную информацию. Например, поле кода категории можно изменить так, чтобы в нем могли храниться только значения в диапазоне от 0 до 15. Кроме того, поле контрольной суммы следует исключить из ключа, так как оно не содержит дополнительной информации; это избыточные данные, намеренно введенные с целью контроля. Для сжатия полей ключа применяются различные поразрядные операции

В хеш-функциях часто используется оператор вычисления остатка (%) с размером таблицы. Вы уже знаете, что выбор простого числа в качестве размера таблицы играет важную роль при квадратичном пробировании и двойном хешировании. Но если сами ключи имеют неслучайное распределение, размер таблицы должен быть простым числом независимо от выбора системы хеширования. Дело в том, что если многие ключи имеют общий делитель с размером массива, они часто хешируются в одну позицию, а это приводит к группировке. Простой размер таблицы исключает такую возможность.

Выбор хеш-функции

Например, если размер таблицы был бы кратен 50 в нашем примере с деталями машин, то все коды категорий хешировались бы в индексы, кратные 50. Простое число (например, 53) гарантирует, что ключи не будут делиться нацело на размер таблицы. Итак, тщательно анализируйте ключи и адаптируйте алгоритм хеширования для устранения любых аномалий в распределении ключей.

Другая разумная хеш-функция основана на разбиении ключа на группы цифр с последующим суммированием групп. Такое решение гарантирует, что хеш-код будет зависеть от каждой цифры исходных данных. Количество цифр в группе должно соответствовать размеру массива. Другими словами, для массива из 1000 элементов каждая группа должна состоять из трех цифр

Выбор хеш-функции

Предположим, вы хотите хешировать 9-разрядные номера социального страхования для линейного пробирования. Если размер массива равен 1000, число из 9 цифр делится на три группы из трех цифр. Так, для кода 123-45-6789 вычисляется ключ 123 + 456 + 789 = 1368. Оператор % усекает полученную сумму, чтобы максимальное значение индекса составляло 999. В нашем примере 1368%1000 = 368. Если бы размер массива был равен 100, то ключ из 9 цифр пришлось бы разделить на четыре группы из двух цифр и одну группу из одной цифры: 12 + 34 + 56 + 78 + 9 = 189, и 189%100 = 89. Когда размер массива кратен 10, работу этой схемы легко понять. Но как было показано для других хеш-функций, оптимальный размер массива должен быть простым числом.

Выводы

Хеш-таблица создается на базе массива.

Диапазон значений ключей обычно больше размера массива.

Хеш-функция преобразует значение ключа в индекс массива.

Типичным примером базы данных, эффективно реализуемой в форме хештаблицы, является словарь английского языка.

Хеширование ключа в уже заполненную ячейку массива называется коллизией.

Существует две основные схемы разрешения коллизий: открытая адресация и метод цепочек.

При открытой адресации элементы данных, хешируемые в заполненную ячейку массива, размещаются в другой ячейке.

В методе цепочек каждый элемент массива содержит связанный список. Все элементы данных, хешируемые в заданный индекс массива, вставляются в этот список.

Выводы

В настоящей главе были рассмотрены три разновидности открытой адресации: линейное пробирование, квадратичное пробирование и двойное хеширование.

При линейном пробировании смещение всегда равно единице. Таким образом, если вычисленный хеш-функцией индекс массива равен x, то пробирование переходит к ячейкам x, x + 1, x + 2, x + 3 и т. Д

Количество шагов, необходимых для обнаружения элемента, называется длиной пробирования.

При линейном пробировании в хеш-таблице появляются непрерывные последовательности заполненных ячеек, называемые первичными группами. Наличие таких групп снижает эффективность хеширования.

При квадратичном пробировании смещение x равно квадрату номера шага. Таким образом, процесс пробирования проверяет ячейки x, x + 1, x + 4, x + 9, x + 16 и т. д.

Выводы

Квадратичное пробирование решает проблему первичной группировки, но не избавляет от (менее опасной) вторичной груп

Вторичная группировка возникает из-за того, что для всех ключей, хешируемых в одно значение, в процессе перебора применяется одна и та же последовательность смещений.

Все ключи, хешируемые в одно значение, следуют по единой последовательности проб, потому что величина смещения не зависит от ключа.пировки.

При двойном хешировании величина смещения зависит от ключа, а для его вычисления используется вторичная хеш-функция.

Если вторичная хеш-функция возвращает значение s, то процесс пробирования проверяет ячейки x, x + s, x + 2s, x + 3s, x + 4s и т. д. Значение s зависит от ключа, но остается неизменным в процессе пробирования.

Коэффициентом заполнения называется отношение количества элементов данных в хеш-таблице к размеру массива.

Выводы

Максимальный коэффициент заполнения при открытой адресации должен оставаться равным примерно 0,5. Для двойного хеширования с таким коэффициентом заполнения средняя длина пробирования при поиске равна 2.

Когда коэффициент заполнения при открытой адресации приближается к значению 1,0, время поиска стремится к бесконечности.

При открытой адресации очень важно, чтобы хеш-таблица не заполнялась выше определенного порога.

Для метода цепочек приемлем коэффициент заполнения 1,0.

При таком коэффициенте заполнения успешный поиск в среднем требует 1,5 пробы, а безуспешный поиск — 2,0 пробы.

Длина пробирования при реализации метода цепочек возрастает линейно с коэффициентом заполнения.

Выводы

Хеширование строки может осуществляться умножением кодов символов на разные степени констант, суммированием произведений и применением оператора вычисления остатка (%) для сокращения результата до размера хештаблицы.

Чтобы избежать переполнения, можно применять оператор % на каждом шаге процесса (представление многочлена по методу Горнера).

Размер хеш-таблицы обычно должен быть простым числом. Это особенно важно при квадратичном пробировании и методе цепочек.

Хеш-таблицы могут использоваться для организации внешнего хранения данных. Одно из возможных решений — хеш-таблица, в которой хранятся номера блоков дискового файла