ЛАБОРАТОРНАЯ РАБОТА ХЕШИРОВАНИЕ

Цель работы: изучить основные методы организации таблиц идентификаторов, получить представление о преимуществах и недостатках, присущих различным методам организации таблиц идентификаторов.

**Теоретические сведения**

**Хеширование - общие понятия**

Эффективными методами поиска являются методы, минимизирующие число сравнений. В идеале хотелось бы иметь такую организацию данных, при которой вообще не было бы ненужных сравнений.

Если каждый ключ должен быть извлечен за одно обращение, то положение записи (ее адрес) должно зависеть только от значения ключа этой записи. Следовательно, необходим метод преобразования ключа в некоторый адрес в заданном диапазоне.

**Хеш-таблица** – это организация данных (*в виде таблицы*), при которой адрес каждой записи ***а*** этой таблицы определяется как значение некоторой функции ***h(k),*** называемой **хеш-функцией**. Здесь ***k*** — значение ключа записи ***а***.

Если число записей невелико и заранее известно, то можно построить функцию преобразования заданного множества ключей в различные адреса. Если возможно, то в последовательные адреса, что выгодно при использовании виртуальной памяти. Если же число записей велико, то найти такую функцию оказывается сложно. В случае если число различных значений ключей, вероятность появления которых отлична от нуля, превышает размер хеш-таблицы, построение функции оказывается невозможным. В этом случае приходится отказываться от идеи однозначности и рассматривать хеш-функцию как функцию, рассеивающую множество ключей во множестве адресов **О(*m — 1)*.**

Любая функция хеширования должна как можно равномернее распределять ключи по всему диапазону значений адресов. Однако, всегда существует вероятность того, что найдутся различные ключи k1 ≠ k2, при этом значение хеш-функции может совпадать: h(k1) = h(k2).). Такая ситуация называется к**оллизией** при хешировании.

Ключи k1 и k2 называются **синонимами** хеш-функции h, если h(k1) = h(k2).

Для метода хеширования главными задачами являются выбор хеш-функции и нахождение способа разрешения возникающих коллизий.

### Организация хеширования

Процедура преобразования ключа в адрес выполняется в три этапа.

1. Если ключ не цифровой (текстовое значение), он преобразуется в соответствующее цифровое представление таким образом, чтобы исключить потерю информации, содержащейся в ключе.

Например, буквы переводятся в цифровой код (ASCII-таблица), символы представляются в виде строки битов.

1. Ключи в цифровом (или битовом) представлении затем преобразуются функцией хеширования в совокупность произвольно распределенных чисел, значения которых имеют тот же порядок, что и значения адресов основной области памяти.

*Желательно, чтобы набор ключей был распределен как можно равномернее в диапазоне допустимых адресов.*

1. Полученные числа умножаются на константу, что позволяет разместить их строго в диапазоне значений адресов основной области памяти.

Например, в результате выполнения этапа II значения - это четырехзначные числа [0 .. 9999] а в области памяти выделено всего 7000 адресов; полученные числа следует домножить на 0.7, что позволит получать адреса в интервале [0 .. 6999].

### Хеш-функции

При выборе хеш-функции следует учитывать сложность ее вычисления, а также равномерность распределения значений, которая позволяет не только сократить число коллизий, но и не допустить “скопления” значений в отдельных частях таблицы.

Для каждого конкретного множества возможных ключей можно “подобрать”, свою, возможно наилучшую, хеш-функцию распределения ключей по таблице. Но существуют и универсальные хеш-функции, дающие хорошие результаты в большинстве случаев. Далее будут рассмотрены основные хеш-функции, используемые на этапе II для преобразования ключа в адрес.

### Метод деления

Наиболее распространенная функция хеширования основывается на методе деления и определяется в виде

**H(K) = (K mod m) + 1**,   
где **K** - ключ, **mod** - операция, вычисляющая остаток от деления, m - делитель.

Равномерность распределения получаемых адресов во многом зависит от выбранного делителя **m**. Следует избегать четных делителей, так как при этом четные и нечетные ключи отображаются соответственно в четные и нечетные адреса. Если множество ключей состоит в основном из четных или в основном из нечетных ключей, будут возникать многочисленные коллизии.

Если **m** является большим простым числом, то количество коллизий невелико. Также неплохие результаты дает выбор в качестве делителя **m** числа, которое не является простым и не содержит простых сомножителей, меньших 20.

### Метод середины квадратов

При хешировании по методу середины квадратов сначала ключ умножается сам на себя. В качестве адреса выбирается столько цифр из середины результата, какова требуемая длина адреса.

Пример. Пусть дан ключ 234583. При возведении его в квадрат получаем 55029183889. Если требуется 100 адресов, то адрес будет равен 91 (5502**91**83889), если необходимо 1000 адресов (5502**918**3889) - 918, если 10000 (550**2918**3889) - 2918 и т.д.

Если требуется получить количество адресов, некратное 10, например 736. В этом случае необходимо взять три средние цифры и умножить на коэффициент 0.736. Например 918\*0.736 = 676.

Исследования с реальными данными показали, что метод середины квадратов дает неплохой результат при условии, что ключи не содержат много левых или правых нулей подряд.

### Метод свертывания

В методе свертывания ключ разбивается на части, каждая из которых имеет длину, равную длине требуемого адреса. Адрес формируется как сумма этих частей; при этом перенос в старший разряд игнорируется. И как показывает практика, разбиение справа налево предпочтительнее разбиения слева направо.

Пример. Пусть дан ключ 3415768898. Для двух-, трех- и четырехцифрового адреса получим следующие значения:

|  |  |
| --- | --- |
| **Количество адресов** | **Значения** |
| 100 | 34+15+76+88+98 = 11 |
| 1000 | 3+415+768+898 = 084 |
| 10000 | 34+1576+8898 = 0508 |

Метод свертывания используется, обычно, для больших ключей.

### Метод преобразования системы счисления

В основе метода лежит преобразование значения ключа **k**, выраженного в системе счисления с основанием **р (k = a0 + а1р + a2p2 + ...)**, в систему счисления с основанием **q (h(k) = a0 + а1q + a2q2 + ...)** при условии, что **р < q**.

Трудоемкость (число операций) этого метода оказывается большей, чем методов деления или умножения.

### Метод деления многочленов

Пусть **k**, выраженное в двоичной системе счисления, записывается как

**k = 2nbn + ... + 2b1 + b0**, и пусть размер хеш-таблицы **m** является степенью двойки **m = 2Р.** Представим двоичный ключ **k** в виде многочлена вида

**k(t) = bntn + ... + b1t+ b0.**

Определим остаток от деления этого многочлена на постоянный многочлен вида **c(t) = tm+cm-1tm-1+…c1t+c0**.

Этот остаток, рассматриваемый в двоичной системе счисления, используется в качестве значения хеш-функции **h(k**). Для вычисления остатка от деления многочленов используют полиномиальную арифметику по модулю 2. Если в качестве **c(t)** выбрать простой неприводимый многочлен, то при условии близких, но не равных **k1** и **k2**, обязательно будет выполняться условие

**h(k1) ≠ h(k2).**

Многочлен c(t) называется простым неприводимым многочле­ном, если его нельзя представить в виде произведения **c(t) = q(t) x r(t), где q(t) и r(t)** — многочлены, отличные от константы. Эта функция обладает сильным свойством рассеивания скученностей.

### Метод умножения

Представим значение ключа **k** в виде двоичного числа и примем размер хеш-таблицы **m** равным **2 Р**. Умножим дробь **d** на k и возьмем дробную часть числа, которую обозначим как {**k x d}**, а в качестве значения хеш-функции используем **p** старших разрядов **1**, т. е. **h(k) = ⎣m x {kd}⎦**, где **⎣x⎦** — наибольшее целое число, не превосходящее **х**.

Рекомендуется в качестве значения d брать иррациональное число, например золотое сечение (**sqrt(5) -1)/2**. При **d =1/m**  метод эквивалентен методу деления.

### Коллизии

Существует два основных метода разрешения коллизий: метод открытой адресации и метод цепочек. Рассмотрим оба метода при выполнении основных операций (Поиск, Добавление, Удаление ключа) в хеш-таблицах.

### Метод открытой адресации

**Поиск и удаления ключа**

Алгоритм поиска ключа состоит в вычислении его хеш-адреса и просмотре той же последовательности элементов. Либо, при удачном поиске, будет найден искомый ключ, либо, при поиске неудачном, будет найден пустой элемент или будут просмотрены все элементы массива.

Если из массива удаляется ключ Ki, то в соответствующий элемент массива необходимо занести специальный признак удаленного ключа (например, отрицательное число, если все ключи положительные, или наибольшее положительное число), т.е. значение, которое не равно ни одному возможному значению ключа.

При поиске помеченные как удаленные записи рассматриваются как занятые, а при занесении, как свободные.

Пример применения метода открытой адресации.

Пусть задан массив S из 11 элементов, хеш-функция H(K) = (K mod 11) + 1 и последовательность ключей 88, 21, 96, 86, 11, 22, 5, 29, 19. Результаты заполнения массива методом открытой адресации показаны на рисунке.

Как видно из рисунка, коллизия возникает при занесении ключей 11, 22 и 19.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **88** | **11** | **22** | **19** |  | **5** |  | **29** | **96** | **86** | **21** |

**Анализ метода**

Разрешение коллизий с помощью метода открытой адресации имеет ряд недостатков. Основным недостатком метода является эффект скучивания, который значительно усиливается при почти заполненном массиве.

Эффект скучивания заключается в следующем. Для рассмотренного выше примера при пустом массиве вероятность того, что новый элемент попадет в первую позицию, равна 1/11. Пусть теперь первая позиция занята. При втором добавлении вероятность того, что будет занята позиция два, в два раза больше, чем вероятность попадания в остальные позиции и т.д.

Следовательно, имеет место тенденция возникновения все более длинных последовательностей занятых подряд позиций, что увеличивает как время поиска, так и время добавления элементов.

Кроме того, так как в качестве структуры хранения используется массив, т.е. структура данных с фиксированным количеством элементов, то всегда существует вероятность переполнения.

### Метод цепочек

В этом методе для разрешения коллизий в каждую запись хеш-таблицы добавляется указатель для поддержания связанного списка. Сами списки могут размещаться как в памяти, принадлежащей хеш-таблице (внутренние цепочки), так и в отдельной памяти (внешние цепочки).

### Метод внешних цепочек

Хеш-таблица представляет собой массив связанных списков размера m (элементы массива обычно имеют индексы от 0 до m - 1). После вычисления значения хеш-функции а = h(k) задача сводится к последовательному поиску в списке.

Возможно, что список размещается во внешней памяти. В этом случае для ускорения поиска желательно, чтобы записи с ключами-синонимами при вставке попадали в один кластер файла. Если число синонимов становится слишком большим, можно вместо линейных списков использовать дерево поиска.

Анализ: метод цепочек лишен большинства недостатков, присущих методу открытой адресации. Так, использование динамических списков практически не ограничивает количества добавляемых элементов.

Удаление элемента состоит в исключении узла из связного списка, что никак не влияет на эффективность как алгоритма поиска, так и алгоритма включения. Основным недостатком метода цепочек является то, что для указателей требуется дополнительная память. Кроме того, если списки станут слишком длинными, то теряет смысл вся идея хеширования.

### 

### Внутренние цепочки

В методе внутренних цепочек связанные списки для синонимов поддерживаются внутри таблицы. Для поиска свободного места в таблице можно использовать разные методы. Например, последовательный просмотр позиций таблицы.

При вставке новых ключей проявляется тенденция объединения хеш-адресов в группы, что приводит к срастанию списков. Таким образом, в один список могут попасть ключи, не являющиеся синонимами по хеш-функции, удлиняя список. Часто метод внутренних цепочек называют методом срастающихся цепочек.

## Задание на лабораторную работу:

* + - 1. Программно реализовать поиск с использованием хеширования:
* Разработать **две** хэш-функции, предусмотреть обработку коллизий. Хеш–функции и метод для разрешения коллизий выбрать любые из указанных в теоретической части.
* *Захэшировать* входные данные; входные данные – массив записей. Структура записей по вариантам используется из лабораторной работы «Алгоритмы внутренней сортировки».   
  Оценить реализованные хэш-функции на число коллизий в исходном наборе данных и далее использовать хэш-функцию, которая минимизирует коллизии.
* На основании полученных хэш-значений построить хэш-таблицу.
* Реализовать Поиск, Добавление, Удаление элементов.

1. Подготовить отчет.   
   В отчете дополнительно\* описать и обосновать выбор хеш-функций, проанализировать реализованный метод борьбы с коллизиями.

\* дополнительно к описанию структур данных, работе программы.

1. Защитить работу.

**Примеры оформления отчета**

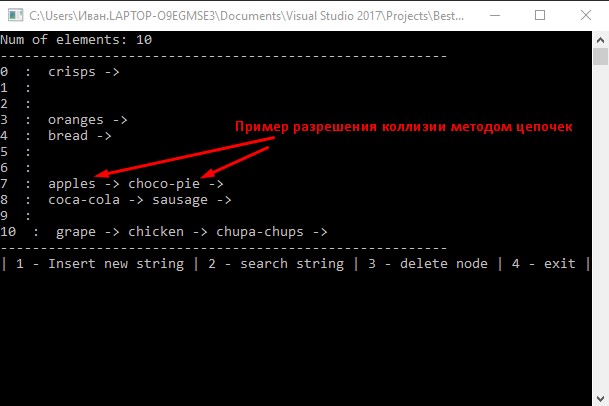


Рисунок 1- Пример вывода хэш-таблицы

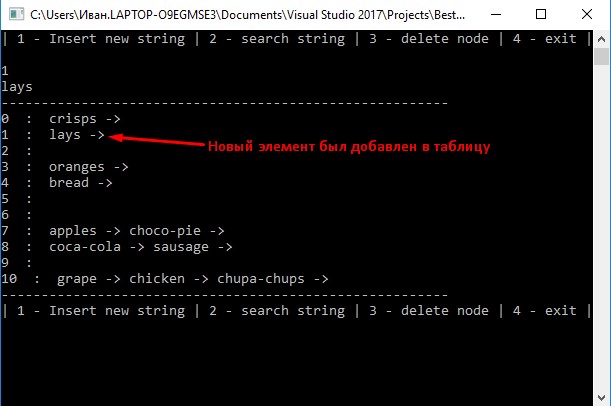


Рисунок 2 - Пример вывода обновленной хэш-таблицы после добавления элемента

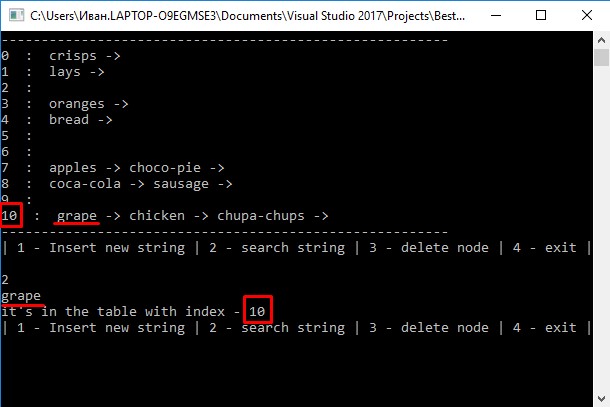


Рисунок 3 - Пример работы поиска по таблице

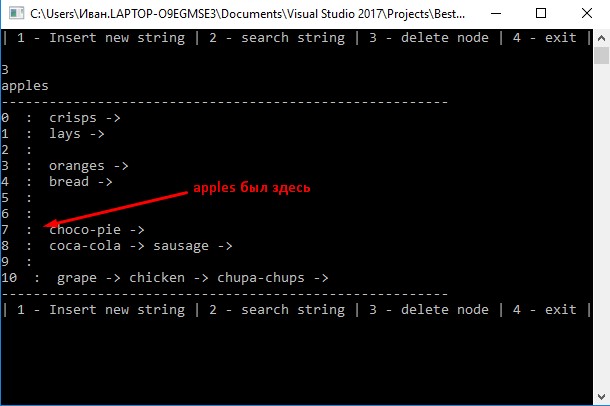


Рисунок 4 - Пример удаления элемента из хэш-таблицы

## Вопросы для самопроверки

1. В чем заключается метод поиска - хеширование?
2. Перечислите основные хеш - функции?
3. В чем суть каждой хеш - функции?
4. Что такое коллизия?
5. Какие методы для разрешения коллизий вы знаете?
6. В чем заключается суть каждого метода для разрешения коллизии?

# Список использованной литературы

1. Валиуллова Н.А. Учебное пособие по курсу "Структуры и алгоритмы обработки данных", Кузнецкий институт информационных и управленческих технологий, Кузнецк, 2008. – 71с.
2. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. — СПб.: Питер, 2017. — 288 с.