МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Брянский государственный технический университет

Утверждаю
Ректор университета

_____О.Н. Федонин

«____»_____2020 г.

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

ХЕШИРОВАНИЕ. ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ РАЗЛИЧНЫХ АЛГОРИТМОВ

Методические указания
к выполнению лабораторной работы №13
для студентов очной, очно-заочной и заочной форм обучения
по направлениям подготовки
230100 «Информатика и вычислительная техника»,
010500 «Математическое обеспечение и администрирование информационных систем»,
231000 «Программная инженерия»

Брянск 2020

УДК 006.91

Структуры и алгоритмы обработки данных. Хеширование. Исследование эффективности различных алгоритмов [Электронный ресурс]: методические указания к выполнению лабораторной работы №13 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки 230100 «Информатика и вычислительная техника», 010500 «Математическое обеспечение и администрирование информационных систем», 231000 «Программная инженерия». – Брянск: БГТУ, 2020.– 26 с.

Разработали:
канд. техн. наук, проф.
В.К. Гулаков
канд. техн. наук, доц.
А.О. Трубаков
канд. техн. наук, доц.
Е.О. Трубаков

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №1 от 13.09.20)

1. ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Целью лабораторной работы является приобретение практических навыков разработки алгоритмов открытого и закрытого хеширования.

Продолжительность лабораторной работы – 4 часа.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Основные понятия, обзор и анализ методов хеширования

Эффективность поиска зависит от структуры данных.

До сих пор мы рассматривали методы поиска, основанные на сравнении данного аргумента K с имеющимися в таблице ключами или на использовании его цифр для управления процессом разветвления.

В случае последовательной структуры данных гарантированно просматривается O(n) элементов, в случае бинарных поисковых деревьев и при бинарном поиске обеспечивается более высокая эффективность $O(\log_2 n)$, т.е. сложность алгоритма поиска зависит от размера таблицы.

В идеале нам хотелось бы выбирать данные за время O(1), когда число необходимых сравнений не зависит от количества элементов данных. Например, выборка элемента осуществляется за время O(1) при использовании в качестве индекса в массиве некоторого ключа.

Основная идея ассоциативной адресации состоит в том, чтобы рассматривать адрес как функцию от ключа. Требуется, чтобы этот адрес вычислялся как можно проще и как можно быстрее.

Хеширование полезно, когда широкий диапазон возможных значений ключа должен быть сохранен в малом объеме памяти, и нужен способ быстрого доступа. Хеш-таблицы часто применяются в базах данных, и, особенно, в языковых процессорах типа компиляторов и ассемблеров, где они изящно обслуживают таблицы идентификаторов. В таких приложениях хеш-таблица - наилучшая структура данных.

Хеш-таблица - это обычный одномерный массив с необычной адресацией, задаваемой хеш-функцией. Размер хеш-таблицы m

должен быть достаточно большим, чтобы в ней оставалось разумно большое число пустых мест. В случае, когда в результате хеширования ключи претендуют на один и тот же адрес, приходится отказываться от идеи однозначности.

Отказ от требования взаимно однозначного соответствия между ключом и адресом означает, что для двух различных ключей $k1 \neq k2$ значение хеш-функции может совпадать: $h(k1) = h(k2)$. Такая ситуация называется коллизией.

Если число записей n невелико и заранее известно, то можно построить идеальную хеш-функцию, без коллизий. Это называется идеальным или совершенным хешированием

Если же число записей велико, то найти такую функцию очень сложно.

Для метода хеширования главными задачами являются:

выбор хеш-функции h так, чтобы уменьшить число коллизий;

нахождение способа разрешения возникающих коллизий, т.е. где разместить конфликтные записи - внутри самого массива или вне его.

Мерой использования памяти в таблицах с прямым доступом служит коэффициент заполненности, определяемый как отношение числа записей к числу мест в таблице $\alpha = n/m$.

Методы хеширования являются самыми быстрыми из известных методов поиска, но имеют три недостатка. Табличный порядок имен обычно не связан с их естественным порядком. Худший случай может оказаться хуже, чем при последовательном поиске. Таблицы, основанные на вычислении адреса, расширить динамически непросто: это может приводить к потере памяти, если таблица слишком велика, или к малой производительности, если таблица слишком мала.

Вне зависимости от используемого метода хеширования сама процедура занесения элемента делится на три этапа:

вычисление хеш-адреса (по хеш-функции);

уточнение хеш-адреса (в случае коллизии);

размещение ключа (и/или элемента).

Имеется много разных хеш-функций, каждая со своими преимуществами и недостатками в зависимости от набора хешируемых ключей.

Требования при выборе хеш-функции:

- минимальная сложность ее вычисления;
- равномерность распределения значений хеш-функции по хеш-таблице, что позволяет:
- сократить число коллизий;
- не допустить скучивания значений ключей в отдельных частях таблицы.

Если ключи не являются числами, то они должны быть преобразованы в целые числа перед применением хеш-функций. Хеш-функция не может вычислять адреса больше чем размер хеш-таблицы, причем для всех ключей K $0<h(K)<m$, где m – размер хеш-таблицы.

Различают три класса хеш-функций:

Идеальные

Универсальные

Специальные

Для каждого конкретного множества возможных ключей можно подобрать свою, возможно наилучшую, хеш-функцию распределения ключей по таблице.

Если содержание таблицы строго определено, число хешируемых ключей невелико, то можно найти идеальную хеш-функцию, но любой новый ключ может ухудшить эту функцию.

Класс универсальных хеш-функций используется в большинстве случаев, если они позволяют равномерно распределять ключи по хеш-таблице.

Если трудно подобрать эффективную универсальную хеш-функцию, то применяются специальные хеш-функции, зависящие от распределения ключей

Идеальной (perfect hash function) или «минимальной совершенной» хеш-функцией называется такая хеш функция, когда:

не возникает коллизий;

размер хеш таблицы и количество ключей одинаково

Это идеальный случай с точки зрения потребления памяти и скорости работы. Однако, поиск таких функций – очень нетривиальная задача.

Совершенное хеширование – это когда нет коллизий, но размер хеш-таблицы несколько больше количества хешируемых ключей

Основные алгоритмы поиска идеальных хеш-функций

- Метод проб и ошибок, предложенный Дж. Чикелли

- Метод обратных чисел, разработанный Дж. Джеске
- Алгоритм, основанный на случайных графах, рассмотренный С. Ботело и другие

Алгоритм метода проб и ошибок

- Вначале упорядочиваем буквы по частоте их использования в словах и присваиваем им коды 0,1,2, ... начиная с буквы, обладающей большей частотностью.
- Затем вычисляем значения хеш-функции для каждого ключа, используя числовые коды букв, и упорядочиваем ключи по значениям *h(k).*
- В случае коллизий исследуем, не может ли функция стать минимально совершенной, если изменить коды букв, обладающих малой частотностью.
- Для слов, содержащих буквы с частотностью 1, за счет изменения кодов этих букв удается довольно свободно изменять *h(k).* Корректируя коды букв невысокой частотности, можно довольно успешно избежать коллизий.

При хешировании для вычисления адреса используются много различных **универсальных хеш – функций**, каждая из которых имеет свои преимущества и недостатки. Наиболее известные хеш-функции:

метод деления;
метод умножения;
метод "середины квадрата";
метод свертки (слияния);
метод преобразования системы счисления;
метод деления многочленов;
хеш-функция для строковых ключей,
и другие.

- На практике метод используется в большинстве приложений, работающих с хешированием и требует двух шагов:

в начале ключ должен быть преобразован в целое число;
затем полученное значение вписывается в диапазон 0...m-1 (m-размер хеш-таблицы) с помощью оператора получения остатка.

- H=mod(k, m),

где k- ключ, m-размер хеш-таблицы.

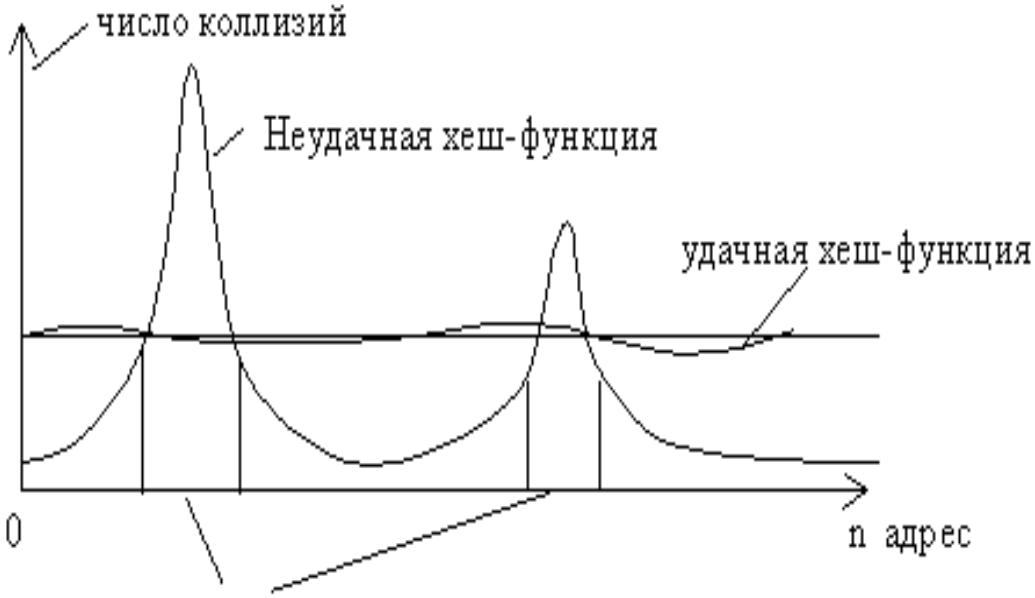
Все рассмотренные до сих пор функции хеширования не зависят от закона распределения значений ключей. Функции хеширования, зависящие от закона распределения ключей, отличны от универсальных. Тогда для заданного ключа x зависящая от распределения ключей функция хеширования H, определяется выражением

H (x) =⌈ mF(x)⌉.

Однако в большинстве случаев функция распределения F неизвестна и должна быть аппроксимирована. Все зависящие от закона распределения ключей функции хеширования различаются лишь подходом, используемым для оценки F.

Оценка качества хеш-функции

Для того чтобы предварительно оценить качество хеш-функции можно провести имитационное моделирование. Моделирование проводится следующим образом. Формируется целочисленный массив, длина которого совпадает с длиной хеш-таблицы. Случайно генерируется достаточно большое число ключей, для каждого ключа вычисляется хеш-функция. В элементах массива просчитывается число генераций данного адреса. По результатам такого моделирования можно построить график распределения значений хеш-функции.



Области концентрации коллизий

Традиционно рассматриваются две классические стратегии преодоления коллизий:

открытую адресацию с линейным перебором

метод цепочек.

Возможны различные варианты этих стратегий, которые имеют свои достоинства и недостатки.

Методы открытой адресации. Суть методов - пользуясь каким-либо правилом, обеспечивающим перебор всех элементов, просто просматривать один за другим элементы хеш-таблицы.

Наиболее распространенными схемами открытой адресации являются:

линейная открытая адресация (линейное опробование);

случайное опробование;

квадратичная открытая адресация (квадратичное опробование);

открытая адресация с двойным хешированием

Линейное опробование. каждая ячейка таблицы помечена как незанятая. Поэтому при добавлении нового ключа всегда можно определить, занята ли данная ячейка таблицы или нет. Если да, алгоритм осуществляет перебор по кругу, пока не встретится «открытый адрес» (свободное место).

Если размер таблицы велик относительно числа хранимых там ключей, метод работает хорошо, поскольку хеш-функция будет равномерно распределять ключи по всему диапазону и число коллизий будет минимальным.

По мере того как коэффициент заполнения таблицы приближается к 1, эффективность процесса заметно падает.

Квадратичное опробование. Эта схема открытой адресации, при которой $h_i(k)=h_0(k)+c_i+d_i^2$, где c и d — константы. Благодаря нелинейности этой адресации удается избежать увеличения числа проб при числе коллизий более 2. Однако в случае почти заполненной таблицы не удается избежать «вторичного скучивания».

Это связано с тем, что при большом числе ключей коллизии одной группы вступают в коллизию с другими ключами.

С одной стороны, этот метод дает хорошее распределение по таблице, а с другой занимает больше времени для просчета.

Двойное хеширование. Один из способов преодоления *вторичного скучивания* состоит в использовании второй функции хеширования, не зависящей от первой.

Предположим, например, что первой функцией хеширования является h_1, такая, что $h_1(x_1) = h_1(x_2) = i$, где $x_1 \neq x_2$. Теперь, если мы имеем вторую функцию хеширования h_2 такую, что $h_2(x_1) \neq h_2(x_2)$ при $x_1 \neq x_2$, то в качестве значения параметра с можно использовать значение $h_2(x_1)$ или $h_2(x_2)$. Если h_1 и h_2 являются независимыми, две случайные последовательности адресов, сгенерированные с помощью такого метода, будут различными. Следовательно, мы избавляемся от вторичного скучивания. Такая вариация метода открытой адресации называется *двойным хешированием.*

Метод цепочек. Хеш-таблица рассматривается как массив связанных списков или деревьев. Каждый такой список называется **блоком** (bucket) и содержит записи, отображаемые хеш-функцией в один и тот же табличный адрес. Эта стратегия разрешения коллизий называется **методом цепочек (chaining with separate lists).**

Если таблица является массивом связанных списков, то элемент данных просто вставляется в соответствующий список в качестве нового узла. Чтобы обнаружить элемент данных, нужно применить хеш-функцию для определения нужного связанного списка и выполнить там последовательный поиск. В общем случае метод цепочек быстрее открытой адресации, так как просматривает только те ключи, которые попадают в один и тот же табличный адрес.

Кроме того, открытая адресация предполагает наличие таблицы фиксированного размера, в то время как в методе цепочек элементы таблицы создаются динамически, а длина списка ограничена лишь количеством памяти.

Списки образуются по мере необходимости по одному на каждый возможный хеш-адрес таблицы. Сами списки могут размещаться как в памяти, принадлежащей хеш-таблице, так и в отдельной памяти. В первом случае *цепочки* называются *внутренними,* а во втором случае — *внешними.*

Поиск в хеш таблице. Рядом с адресом в дополнительном поле записывается информация о коллизии.

- Пример

  Последовательность содержит следующие данные:
  54, 77, 94, 89, 14, 45, 76

Размер хешируемой таблицы A равен 11. Для хеширования используем простейшую хеш-функцию h(key)=mod(key,11). Для преодоления коллизии используется линейное опробование

Получаем следующую хеш таблицу

	0 ,1	1,1	2,1	3,1	4,0	5,0	6,0	7,0	8,0	9,0	10,1
A [	77	89	45	14	76		94				54]

Для того чтобы найти ключ 76, надо сделать 5 попыток последовательного просмотра ячеек.

- при поиске открытым хешированием переход от адреса к адресу осуществляется по признаку коллизии
- При поиске с использованием метода цепочек информация о коллизиях не требуется, а просто пробегается соответствующая цепочка

Удаление элементов хеш-таблицы. Обрабатывать удаление можно, помечая элемент как удаленный, а не как пустой. Таким образом, каждая ячейка в таблице будет содержать уже одно из трех значений: пустая, занятая, удаленная. При поиске удаленные элементы будут трактоваться как занятые, а при вставке – как пустые, соответственно.

Другие методы хеширования

Линейное хеширование (linear hashing, LH)

Линейное хеширование с частичной экспансией(Larson, 1982).

Виртуальное хеширование ((virtual hashing, VH, Litwin, 1978),

Динамическое хеширование (dynamic hashing, DH)

Расширяемое хеширование (extendible hashing)

Trie хеширование. При использовании этого подхода индекс не содержит всех символов ключа, а только его часть.

Спиральное хеширование (Martin, 1979).

Многомерное хеширование

MOLHPE - линейное хеширование при многомерном поиске

Z-Hashing

Quantile Hashing

Разновидности расширяемого хеширования при многомерном поиске

EXCELL

Grid File (R-File, Twin Grid File, BANG File)

Хеширование по сигнатуре.

Одностороннее хеширование

3. ТЕХНИКА БЕЗОПАСНОСТИ ПРИ ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Меры безопасности при работе с электротехническими устройствами соответствуют мерам безопасности, принимаемым при эксплуатации установок с напряжением до 1000 В и разработанным в соответствии с «Правилами техники безопасности при эксплуатации электроустановок потребителей», утвержденными Главгосэнергонадзором 21 декабря 1984 г.

Студенту не разрешается приступать к выполнению лабораторной работы, если замечены какие-либо неисправности в лабораторном оборудовании.

Студент не должен прикасаться к токоведущим элементам электрооборудования, освещения и электропроводке, открывать двери электрошкафов и корпусов системных блоков, мониторов.

Студенту запрещается прикасаться к неизолированным или поврежденным проводам и электрическим устройствам, наступать на переносные электрические провода, лежащие на полу, самостоятельно ремонтировать электрооборудование и инструмент.

Обо всех замеченных неисправностях электрооборудования студент должен немедленно поставить в известность преподавателя или лаборанта.

При выполнении лабораторной работы необходимо соблюдать осторожность и помнить, что только человек, относящийся серьезно к своей безопасности, может быть застрахован от несчастного случая.

4. ЗАДАЧИ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Составьте хеш-таблицу, содержащую буквы и число их вхождений во введенной строке. Вывести таблицу на экран. Осуществить поиск введенной буквы в хеш-таблице.

2. Постройте хеш-таблицу из слов произвольного текстового файла, задав ее размерность с экрана. Выведите построенную таблицу слов на экран. Осуществите поиск введенного слова. Выполните программу для различных размерностей таблицы и сравните число сравнений. Удалите все слова, начинающиеся на указанную букву, выведите таблицу.

3. Постройте хеш-таблицу для зарезервированных слов, используемого языка программирования (не менее 20 слов), содержащую *HELP* для каждого слова. Выдайте на экран подсказку по введенному слову. Добавьте подсказку по вновь введенному слову, используя при необходимости реструктуризацию таблицы. Сравните эффективность добавления ключа в таблицу или ее реструктуризацию для различной степени заполненности таблицы.

4. В текстовом файле содержатся целые числа. Постройте хеш-таблицу из чисел файла. Осуществите поиск введенного целого числа в хеш-таблице. Сравните результаты числа сравнений при различном наборе данных в файле.

5. Провести сравнение скорости поиска в массиве с помощью хеширования и бинарного дерева поиска.

6. В системе бронирования авиационных билетов имеется следующая информация:

1) перечень рейсов;

2) список пассажиров на каждый рейс;

3) очередь пассажиров, в случае когда число заявок на билеты превышает количество мест.

Организовать в основной памяти хешированный список пассажиров. Выдать по имени пассажира полную информацию обо всех списках и очередях, в которых он находится. Считать, что величина каждой очереди ограничена количеством мест на соответствующем рейсе.

7. В файле записаны фамилии студентов и их анкетные данные. Организовать на основе хеширования вспомогательный файл, обеспечивающий быстрый поиск информации по указанному студенту.

8. В подсистеме аннулирования заказов на авиационные билеты имеется следующая информация:

а) перечень рейсов;

б) список пассажиров на каждый рейс;

в) очередь пассажиров, в случае, когда число заявок на билеты превышает число мест.

Аннулирование ведет к удалению пассажира из списка на указанный рейс или удаление из соответствующей очереди. Если пассажир удаляется из списка на указанный рейс, и имеется очередь на указанный рейс, то первый в очереди пассажир попадает в список

на указанный рейс. Организовать в основной памяти приведенную информацию. Обеспечить аннулирование заказа с помощью хеш-поиска в списке пассажиров на указанный рейс из соответствующей очереди. Считать, что величина очереди не превышает числа мест на указанный рейс.

9. Составить программу хеширования массива методом цепочек и последующего поиска записи с заданным ключом. Вывести на печать хеш-таблицу и количество сравнений при поиске

10. Напишите алгоритм и программу хеширования с разрешением коллизий методом открытой адресации. Вывести на печать хеш-таблицу и количество сравнений при различных размерах хеш-таблицы.

11. Напишите алгоритм и программу двойного хеширования и последующего поиска записи с заданным ключом.

12. Напишите алгоритм и программу, использующую функции повторного хеширования, зависящую от числа раз повторного хеширования. Вывести на печать исходные данные, хеш-таблицу и количество сравнений при различных размерах хеш-таблицы.

13. Для отображения данных в табличные индексы используйте хеш-функцию $hashf(x) = x \% 11$. Данные вставляются в таблицу в следующем порядке: 11, 13, 12, 34, 38, 33, 27, 22.

а) Постройте хеш-таблицу методом открытой адресации.

б) Постройте хеш-таблицу методом цепочек.

в) Для обоих методов определите коэффициент заполнения, среднее число проб, необходимое для обнаружения элемента в таблице и среднее число проб для констатации отсутствия элемента в таблице.

14. В памяти ЭВМ хранятся списки номеров телефонов и фамилий абонентов, упорядоченных в алфавитном порядке. Составить программу, обеспечивающую быстрый поиск фамилии абонента по номеру телефона.

15. Построить идеальную хеш-таблицу для заданного набора текстовых ключей. Число ключей (и размер таблицы) равен 10. В качестве ключей взять 10 любых служебных слов языка C++. Для преобразования текстовых ключей в числовые значения использовать суммирование кодов символов текстового ключа: код (End) = код (E) + код (n) + код (d). Преобразование числового кода ключа в значение индекса выполнить с помощью простейшей хеш-

функции, которая берет остаток от целочисленного деления кода на размер хеш-таблицы (в задании – 10).

Для вычисления индексов ключей написать вспомогательную программу, которая в диалоге многократно запрашивает исходный текстовый ключ (служебное слово языка C++) и вычисляет значение хеш-функции. Программа должна вычислять длину очередного текстового ключа с помощью функции Length, в цикле вычислять сумму кодов всех символов ключа и брать остаток от деления этой суммы на 10.

Если некоторые исходные ключи будут конфликтовать друг с другом, можно изменить исходное слово, например – сменить регистр начальной буквы или всех букв в слове, полностью заменить слово на близкое по значению (End на Stop и т.д.), ввести какие-либо спецсимволы или придумать другие способы

После подбора неконфликтующих ключей написать основную программу, которая должна:

ввести подобранные ключи и расположить их в ячейках хеш-таблицы в соответствии со значением хеш-функции

вывести хеш-таблицу на экран

организовать циклический поиск разных ключей, как имеющихся в таблице (с выводом местоположения), так и отсутствующих.

16. Реализовать метод внутреннего хеширования. Исходные ключи – любые слова (например – фамилии). Размер хеш-таблицы должен задаваться в программе с помощью константы m. Хеш-функция – такая же, что и в задании 14, но делить надо на константу m. В случае возникновения конфликта при попытке размещения в таблице нового ключа, для него ищется первое свободное по порядку место по формуле

j = ((h (ключ) + i) mod m) + 1, где i = 0, 1, 2, . . . , m-2

Программа должна выполнять следующие действия:

- добавление нового ключа в таблицу с подсчетом сделанных при этом сравнений
- поиск заданного ключа в таблице с подсчетом сделанных при этом сравнений
- вывод текущего состояния таблицы на экран

После отладки программы необходимо выполнить ее для разных соотношений числа исходных ключей и размерности таблицы: взять 10 ключей и разместить их поочередно в таблице размерности 11, 13

и 17. Для каждого случая найти суммарное число сравнений, необходимое для размещения ключей и их поиска. Сделать вывод о влиянии количества пустых мест в таблице на эффективность поиска.

17. Реализовать метод открытого хеширования. Исходные ключи – любые слова (например – фамилии). Размер хеш-таблицы должен задаваться в программе с помощью константы m. Хеш-функция – такая же, что и в задании 15, но делить надо на константу m. В случае возникновения конфликта при попытке размещения в таблице нового ключа этот ключ добавляется в конец вспомогательного списка. Это требует включения в каждую ячейку хеш-таблицы двух указателей на начало и конец вспомогательного списка.

Программа должна выполнять следующие действия:

- добавление нового ключа в таблицу с подсчетом сделанных при этом сравнений
- поиск заданного ключа в таблице с подсчетом сделанных при этом сравнений
- вывод текущего состояния таблицы на экран
- удаление заданного ключа из таблицы

Алгоритм удаления:

- вычислить хеш-функцию и организовать поиск удаляемого элемента в таблице
- если удаляемый элемент найден в ячейке таблицы, то эта ячейка либо становится пустой (если связанный с ней список пуст), либо в нее записывается значение из первого элемента списка с соответствующим изменением указателей
- если удаляемый элемент найден в списке, то производится его удаление с изменением указателей

После отладки программы необходимо выполнить ее для разных соотношений числа исходных ключей и размерности таблицы: взять 20 ключей и разместить их поочередно в таблице размерности 9, 17 и 23. Для каждого случая найти суммарное число сравнений, необходимое для размещения ключей и их поиска. Сделать вывод о влиянии размерности таблицы на эффективность поиска.

18. Таблица содержит данные об успеваемости студентов, в качестве ключа используется фамилии студента. Разработать функции включения, удаления и поиска записи по ключу. Хеш-

таблица с внешними цепочками, строка таблицы содержит первую запись по данному входу и указатель начала внешней цепочки по данному входу;

19. Таблица содержит данные об успеваемости студентов, в качестве ключа используется фамилии студента. Разработать функции включения, удаления и поиска записи по ключу, но строка таблицы содержит только указатель начала внешней цепочки;

20. Таблица содержит данные об успеваемости студентов, в качестве ключа используется фамилии студента. Разработать функции включения, удаления и поиска записи по ключу хеш-таблица с квадратичной открытой адресацией;

21. Таблица содержит данные об успеваемости студентов, в качестве ключа используется фамилии студента. Разработать функции включения, удаления и поиска записи по ключу в хеш-таблице с двойным хешированием.

22. Проверка орфографии. Написать программу, которая проверяла бы правильность введенных слов, используя предложенный словарь. Словарь считывается из текстового файла. Для организации словаря использовать хеш – таблицу. Разрешение коллизий производить методом цепочек. Слова для проверки вводятся с клавиатуры. Если проверяемое слово верно (т.е. есть в словаре), то выводится соответствующее сообщение. Если слово неправильное, то выводятся все возможные варианты его замены. Для вариантов замены использовать список, адресуемый соответствующей строкой хеш-таблицы. Если замен нет, то ничего не выводится.

23. Приведите содержимое хеш-таблицы, образованной в результате вставки элементов с ключами **E A S Y Q U T I O N** в указанном порядке в первоначально пустую таблицу, имевшую начальный размер $M = 4$, которая увеличивается вдвое при ее заполнении наполовину, при разрешении конфликтов методом линейного зондирования. Воспользуйтесь хеш-функцией $\mathbf{11}k \bmod M$ для преобразования k-той буквы алфавита в индекс таблицы.

24. Пусть для хранения множества всех правильных русских слов в программе проверки орфографии используется хеширование. Что нужно добавить, чтобы к тому же уметь находить английский перевод любого правильного слова?

25. Разработайте алгоритм удаления записи из таблицы, если для устранения коллизий используется метод линейного опробования.

26. Составьте алгоритм поиска, основанный на двойном хешировании.

27. Рассмотрите проблемы, которые возникают при формировании алгоритма удаления, если использован метод двойного хеширования.

28. Сформулируйте алгоритм для внешних цепочек, в котором каждый адрес должен быть представлен не односвязной цепочкой, а в виде древовидной структуры.

29. Имеется информация о торговых связях предпринимателей в виде множества пар (A, B) , где A - продавец, а B - покупатель. Общее число предпринимателей может превышать 1000. Часть продавцов является производителями товаров, а остальные перепродают закупленный товар. По новому постановлению, лица, перепродающие товары, объявлены спекулянтами. Требуется на основе хеширования обеспечить быстрый ответ на вопрос, является ли указанный предприниматель спекулянтом.

30. В файле записаны фамилии студентов и их анкетные данные. Организовать на основе хеширования вспомогательный файл, обеспечивающий быстрый поиск информации по указанному студенту.

31. В подсистеме аннулирования заказов на авиационные билеты имеется следующая информация:

1) перечень рейсов;

2) список пассажиров на каждый рейс;

3) очередь пассажиров, в случае когда число заявок на билеты превышает количество мест.

Аннулирование ведет к удалению пассажира из списка на указанный рейс или удаление из соответствующей очереди. Если пассажир удаляется из списка, и имеется очередь на данный рейс, то первый в очереди пассажир попадает в список рейса. Организовать в основной памяти указанную информацию. Обеспечить аннулирование заказа с помощью хеш - поиска в списке пассажиров нужного рейса и соответствующей очереди. Считать, что величина очереди не превышает количества мест на соответствующем рейсе.

32. Составить программу хеширования массива методом цепочек и последующего поиска записи с заданным ключом.

33. Напишите алгоритм и программу хеширования с разрешением коллизий методом открытой адресации.

34. Напишите алгоритм и программу двойного хеширования

35. Напишите алгоритм и программу, использующую функции повторного хеширования, зависящую от числа раз повторного хеширования.

36. Для отображения данных в табличные индексы используйте хеш-функцию hashf(x) = x % 11. Данные вставляются в таблицу в следующем порядке:

11, 13, 12, 34, 38, 33, 27, 22.

а) Постройте хеш-таблицу методом открытой адресации.

б) Постройте хеш-таблицу методом цепочек.

в) Для обоих методов определите коэффициент заполнения, среднее число проб, необходимое для обнаружения элемента в таблице и среднее число проб для констатации отсутствия элемента в таблице.

37. Для отображения данных в табличные индексы используйте хеш-функцию hashf(x) = x % 11. Данные вставляются в таблицу в следующем порядке: 11, 13, 12, 34, 38, 33, 27, 22.

а) Постройте хеш-таблицу методом открытой адресации.

б) Постройте хеш-таблицу методом цепочек.

в) Для обоих методов определите коэффициент заполнения, среднее число проб, необходимое для обнаружения элемента в таблице и среднее число проб для констатации отсутствия элемента в таблице.

38. В памяти ЭВМ хранятся списки номеров телефонов и фамилий абонентов, упорядоченных в алфавитном порядке. Составить программу, обеспечивающую быстрый поиск фамилии абонента по номеру телефона.

5. УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

Каждую задачу решить в соответствии с изученным алгоритмами хеширования данных, реализовав программный код на языке C++. Результаты обработки данных выводить в выходной файл и дублировать вывод на экране.

Каждую задачу реализовать в соответствии с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать метод решения задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке C++;
- разработать контрольный тест к программе;
- отладить программу;
- составить отчет о лабораторной работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каков принцип построения хеш-таблиц?
2. Существуют ли универсальные методы построения хеш-таблиц? Ответ обоснуйте.
3. Почему возможно возникновение коллизий?
4. Каковы методы устранения коллизий? Охарактеризуйте их эффективность в различных ситуациях.
5. Назовите преимущества открытого и закрытого хеширования.
6. Когда поиск в хеш-таблицах становится неэффективен?
7. Как выбирается метод изменения адреса при повторном хешировании?
8. Для чего применяется хеширование.
9. Каким образом подбирается оптимальная хеш-функция.
10. В чем заключается метод хеш-поиска?
11. Для чего используется хеш-функция и какие к ней предъявляются требования?
12. Что такое хеш-таблица и как она используется?

13. Как по трудоемкости соотносятся между собой основные методы поиска (полный перебор, двоичный поиск, хеш-поиск)?
14. Как с помощью простейшей хеш-функции находится расположение в таблице строковых ключей?
15. Какие проблемы могут возникать при построении хеш-таблиц с произвольными наборами ключей?
16. В каких ситуациях можно построить бесконфликтную хеш-таблицу?
17. Где на практике и почему можно использовать бесконфликтные хеш-таблицы?
18. Что такое открытое хеширование и для чего оно применяется?
19. Какие структуры данных используются для реализации открытого хеширования?
20. Какие шаги выполняет алгоритм построения хеш-таблицы при открытом хешировании?
21. Какие шаги выполняет алгоритм поиска в хеш-таблице при открытом хешировании?
22. Какие проблемы могут возникать при использовании открытого хеширования?
23. Как влияет размер хеш-таблицы на эффективность открытого хеширования?
24. Что такое внутреннее хеширование и для чего оно применяется?
25. Какие правила можно использовать для поиска свободных ячеек при внутреннем хешировании?
26. Какие шаги выполняет алгоритм построения хеш-таблицы при внутреннем хешировании?
27. Какие шаги выполняет алгоритм поиска в хеш-таблице при внутреннем хешировании?
28. Как влияет размер хеш-таблицы на эффективность внутреннего хеширования?
29. В каких задачах НЕ следует применять метод хеш-поиска?

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Кнут, Д. Искусство программирования для ЭВМ Т.1, Основные алгоритмы / Д. Кнут – М.: Диалектика Вильямс 2020. – 760 с. ISBN 978-5-907144-23-1

2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт – М.: ДМК Пресс, 2010 – 272 с.: ил. ISBN 978 5 94074 584 6

3. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М: Издательский дом «Вильямс», 2005. –1296 с. ISBN 5-8459-0857-4(рус)

4. Ахо, А. В. Структуры данных и алгоритмы / А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. – М.: Вильямс, 2003. – 382 с. ISBN 10:0-201-00023-7

5. Седжвик, Р. Фундаментальные алгоритмы на C++. Анализ. Структуры данных. Сортировка. Поиск: пер. с англ. /Р. Седжвик. – К.: ДиаСофт, 2001. – 688 с. ISBN 5-93772-054-7

6. Хэзфилд, Р. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: пер. с англ. / Р. Хэзфилд, Л. Кирби [и др.]. – К.: ДиаСофт, 2001. – 736 с. ISBN 966-7393-82-8, 0-672-31896-2

7. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си (+ CD): учеб. Пособие / Б.С. Хусаинов. – М.: Финансы и статистика, 2004. – 464 с. ISBN: 5-279-02775-8

19. Гулаков В. К. Введение в хеширование: монография/ В.К. Гулаков, К. В. Гулаков – Брянск: БГТУ, 2011. – 129 с. ISBN 5-89838-593-4.

Структуры и алгоритмы обработки данных. Хеширование Исследование эффективности различных алгоритмов: методические указания к выполнению лабораторной работы №13 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки: 230100 «Информатика и вычислительная техника», 010500 «Математическое обеспечение и администрирование информационных систем», 231000 «Программная инженерия»

ВАСИЛИЙ КОНСТАНТИНОВИЧ ГУЛАКОВ
АНДРЕЙ ОЛЕГОВИЧ ТРУБАКОВ
ЕВГЕНИЙ ОЛЕГОВИЧ ТРУБАКОВ

Научный редактор В.В. Конкин
Редактор издательства Л.Н. Мажугина
Компьютерный набор В.К. Гулаков

Темплан 2020 г., п. 209

Подписано в печать Формат 1/16
Усл.печ.л. 1,51 Уч.-изд.л. 1,51

Издательство Брянского государственного технического университета
241035, Брянск, бульвар им.50-летия Октября, 7, БГТУ, тел. 58-82-49
Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16