УДК 519.71

МЕТОДОЛОГІЧНИЙ ВИБІР ХЕШ-ФУНКЦІЇ ДЛЯ ОПТИМАЛЬНОЇ СИНХРОНІЗАЦІЇ БАЗИ ДАНИХ НАЦІОНАЛЬНИХ ПАРКІВ ХОРВАТІЇ

Борд Р.В.
Дніпропетровський національний університет
імені Олеся Гончара
Лозовська Л.І.
Національна металургійна академія України

Основні питання, що виникають при пошуку оптимального алгоритму хешування, є дослідження таких властивостей цих алгоритмів як стійкість до пошуку першого та другого прообразів, обчислювальна складність. В роботі проведено аналіз найбільш поширених алгоритмів побудови хеш-функцій. Виявлено їх переваги та недоліки. Визначено основні критерії підбору оптимальної хеш-функції для конкретних практичних задач. Визначено оптимальний алгоритм хешування для конкретної практичної задачі замовника.
Ключові слова: алгоритм хешування, MD4, MD5, SHA, динамічне хешування, метод множення, метод поділу, мінімальне ідеальне хешування.

Постановка проблеми. Дослідження задач алгоритмів хешування є дуже важливим питанням на сьогоднішній день, адже ці задачі є актуальними в практичному значенні. Оскільки всюди, де необхідно перевірити цілісність передачі даних через канали зв'язку, такі як Інтернет, виникає питання перевірки чи дані дійшли від відправника до отримувача у повному обсязі. Це стосується як перевірки цілісності та коректності даних, так і перевірки чи ці дані мають актуальний стан або потребують оновлення.

Основні питання що виникають при пошуку оптимального алгоритму хешування, є дослідження деяких властивостей цих алгоритмів. Такими властивостями є наприклад стійкість до пошуку першого та другого прообразів, обчислювальна складність та інші.

Зокрема, для нашої практичної задачі потрібно було проаналізувати доступні алгоритми хешування, та знайти такий алгоритм, який б водночас був і стійким до пошуку прообразів, але також дуже швидко міг видавати результат хешування, тобто хеш для дуже великих об'ємів вхідних даних. В цій роботі розглянуто найбільш відомі алгоритми хешування, а також представлена реалізація обраного в результаті аналізу алгоритмів, так званого «оптимального» алгоритму для конкретної практичної задачі, яку перед нами поставив замовник.

Аналіз останніх досліджень і публікацій. Процедура хешування інформації застосовується при вирішенні широкого кола різноманітних завдань: в програмуванні при побудові асоціативних масивів, пошуку дублікатів в даних, визначення контрольної суми для захисту при передачі інформації [2].

В останні роки проведено багато досліджень з розробки нових та вдосконалення вже існуючих методів хешування. Наприклад, Лужецький В. А. та Баришев Ю. В. розробили конструкцію багатоканального хешування, що дало можливість узагальнити та удосконалити відомі підходи підвищення стійкості хешування до мультиколізій [3]. Бойко А. О. та Горбенко І. Д. запропонували критерії, показники та методику оцінки функцій хешування з підвищеною швидкодією [4]. При цьому підвищення швидкодії досягається за рахунок розпаралелювання обчислень. Ще одним напрямком є застосування нейронних мереж для вирішення задач захисту інформації [6], включаючи і хешування даних.

Значний внесок у розвиток алгоритмів хешування також внесли Д. Кнут, У. Пітерсон, та Ханс Петер Лун.

Виділення не вирішених раніше частин загальної проблеми. До сьогоднішнього часу дослідження оптимального механізму хешування були слабо представлені, систематичні дослідження у цьому напрямку ще не завершені, а їх важливість не потребує додаткового обґрунтування. Вищезазначене обумовлює необхідність подальших наукових пошуків на сучасному науковому підґрунті.

Мета статті. Метою даної роботи є ознайомлення з основними критеріями підбору оптимальної хеш-функції для конкретних практичних задач, а також визначення оптимального алгоритму хешування для конкретної практичної задачі замовника.

Виклад основного матеріалу. Хеш-функція – це деяка функція h(K), яка бере якийсь ключ K і повертає адресу, за якою проводиться пошук в хеш-таблиці, щоб отримати інформацію, пов'язану з K. Наприклад, K – це номер телефону абонента, а шукана інформація – його ім'я. Функція в даному випадку нам точно визначить, за якою адресою знайти шукане.

Якісна хеш-функція повинна задовольняти двом вимогам:

· її обчислення повинно виконуватися дуже швидко;

· вона повинна мінімізувати кількість колізій.

Отже, перша властивість якісної хеш-функції залежить від комп'ютера, а друга – від даних.

Теоретично неможливо визначити хеш-функцію так, щоб вона створювала випадкові дані з реальних невипадкових файлів. Однак на практиці реально створити досить хорошу імітацію за допомогою простих арифметичних дій. Більш того, часто можна використовувати особливості даних для створення хеш-функцій з мінімальним числом колізій (меншим, ніж при істинно випадкових даних) [3].

Одним з найбільш очевидних і простих способів хешування є метод середини квадрата, коли ключ зводиться в квадрат і береться кілька цифр в середині. Ключ спочатку приводиться до цілого числа, для здійснення з ним арифметичних операцій. Однак такий спосіб добре працює до моменту, коли немає великої кількості нулів зліва чи справа.

Виокремлюють наступні методи хешування:

Метод поділу. При цьому методі використовується залишок від ділення на M [2]:

$$h(K) = K \bmod M.$$

Треба ретельно вибирати цю константу. При парній константі значення функції буде парним при парному K і непарним – при непарному, що призведе до небажаного результату. Також M не повинно бути кратним трьом, оскільки при літерних ключах два з них, що відрізняються тільки перестановкою літер, можуть давати числові значення з різницею, кратній трьом. Наведені міркування наводять на думку, що краще використовувати просте число. У більшості випадків подібний вибір цілком задовільний.

Метод множення (мультиплікативний). Для цього методу хешування [2] використовується формула:

$$h(K) = [M * ((C * K) \bmod 1)]$$

Тут виконується множення ключа на деяку константу C, що лежить в інтервалі [0..1]. Після цього береться дробова частина цього виразу і множиться на деяку константу M, обрану таким чином, щоб результат не вийшов за межі хеш-таблиці. Оператор повертає найбільше ціле, яке менше аргументу. Якщо константа C обрана вірно, то можна домогтися дуже хороших результатів, однак, цей вибір складно зробити. Дональд Кнут зазначає, що множення може іноді виконуватися швидше ділення. Мультиплікативний метод добре використовує те, що реальні файли невипадкові. Наприклад, часто множини ключів є арифметичними прогресіями, коли в файлі містяться ключі {K, K + d, K + 2d, ..., K + td}.

Окремим випадком вибору константи є значення величини золотого перетину $\varphi = (\sqrt{5} - 1) / 2 \approx \approx +0{,}6180339887$. Якщо взяти послідовність {φ}, {2φ}, {3φ},... де оператор {} повертає дробову частина аргументу, то на відрізку [0..1] вона буде розподілена дуже рівномірно. Це явище було вперше помічено Я. Одерфельдом (J. Oderfeld) [2] і доведено С. Сверчковскі (S. Świerczkowski) [2]. У доведені відіграють важливу роль числа Фібоначчі. Стосовно до хешування це означає, що якщо в якості константи вибрати золотий перетин, то функція буде досить добре розсіювати ключі виду {PART1, PART2,..., PARTN}.

Алгоритм MD4. Алгоритм MD4 є більш ранньої розробкою того ж автора Рона Ривеста. Спочатку даний алгоритм був опублікований в жовтні 1990 р, незначно змінена версія була опублікована в RFC 1320 в квітні 1992 р. [3].

Виокремлюють основні цілі алгоритму MD4:

1. *Безпека:* це звичайна вимога до хеш-коду, що полягає в тому, щоб було чисельно неможливо знайти два повідомлення, які мають один і той же дайджест.

2. *Швидкість:* програмна реалізація алгоритму повинна виконуватися досить швидко. Зокрема, алгоритм повинен бути досить швидким на 32-бітної архітектури. Тому алгоритм заснований на простому множині елементарних операцій над 32-бітними словами.

3. *Простота та компактність:* алгоритм повинен бути простим в описі і простим в програмуванні, без великих програм або підстановочних таблиць. Ці характеристики не тільки мають очевидні програмні переваги, але і бажані з точки зору безпеки, тому що для аналізу можливих слабких місць краще мати простий алгоритм.

4. *Бажано little-endian архітектура:* деякі архітектури процесорів зберігають ліві байти слова в позиції молодших адрес байта (little-endian). Інші зберігають праві байти слова в позиції молодших адрес байта (bigendian). Ця відмінність є важливою, коли повідомлення трактується як послідовність 32-бітових слів, тому що ці архітектури мають інверсне уявлення байтів в кожному слові. Ці цілі переслідувалися і при розробці алгоритму MD5 [3]. MD5 є більш складним і, отже, більш повільним при виконанні, ніж MD4. Вважається, що додавання складності виправдовується зростанням рівня безпеки.

Динамічне хешування. Існує техніка, що дозволяє динамічно змінювати розмір хеш-структури. Це – динамічне хешування. Хеш-функція генерує так званий псевдоключ, який використовується лише частково для доступу до елементу. Іншими словами, генерується досить

довга бітова послідовність, яка повинна бути достатньою для адресації всіх потенційно можливих елементів.

Розширюване хешування (extendiblehashing). Розширюване хешування є близьким до динамічного. Цей метод також передбачає зміну розмірів блоків зі зростанням бази даних, але це компенсується оптимальним використанням місця. Замість бінарного дерева розширюване хешування передбачає список, елементи якого посилаються на блоки. Самі ж елементи адресуються за деякою кількістю *і* бітів псевдоключа. При пошуку береться *і* бітів псевдоключа і через список (directory) знаходиться адреса шуканого блоку. Додавання елементів проводиться складніше. Спочатку виконується процедура, аналогічна пошуку. Якщо блок неповний, додається запис в нього і в базу даних. Якщо блок заповнений, він розбивається на два, записи перерозподіляються за описаним вище алгоритмом. У цьому випадку можливе збільшення кількості біт, необхідних для адресації. Таким чином, можлива ситуація, коли кілька елементів показують на один і той же блок. Слід зауважити, що за одну операцію вставки перераховуються значення не більше, ніж одного блоку. Видалення здійснюється за таким же алгоритмом, тільки навпаки.

Отже, основною перевагою розширюваного хешування є висока ефективність, яка не падає при збільшенні розміру бази даних. Крім цього, розумно витрачається місце на пристрої зберігання даних, тому що блоки виділяються тільки під реально існуючі дані, а список покажчиків на блоки має розміри, мінімально необхідні для адресації даного кількості блоків.

Складання хеш-функції – це не вся робота, яку належить виконати програмісту, який реалізує пошук на основі хешування. Крім цього, необхідно реалізувати механізм *розв'язання колізій.* Як і з хеш-функціями існує кілька можливих варіантів, які мають свої переваги і недоліки:

Метод ланцюжків. У разі, коли елемент таблиці з індексом, який повернула хеш-функція, вже зайнятий, до нього приєднується зв'язний список. Таким чином, якщо для кількох різних значень ключа повертається однакове значення хеш-функції, то за цією адресою знаходиться показник на зв'язаний список, який містить всі значення.

Відкрита адресація. Полягає у тому, щоб повністю відмовитися від посилань, просто переглядаючи різні записи таблиці по порядку до тих пір, поки не буде знайдений ключ К або порожня позиція. Ідея полягає у формулюванні правила, згідно з яким за даним ключем визначається «пробна послідовність», тобто послідовність позицій таблиці, які повинні бути переглянуті при вставці або пошуку ключа К. Якщо при пошуку зустрічається порожня клітинка, то можна зробити висновок, що К в таблиці відсутнє, тому що ця клітинка була б зайнята при вставці, тому що алгоритм проходив той самий ланцюжок.

Лінійна адресація використовує циклічну послідовність перевірок і описується наступним алгоритмом [1]:

h(К), h(К – 1), ..., 0, М – 1, М – 2,..., h(К) + 1

Він виконує пошук ключа К в таблиці з М елементів. Якщо таблиця не повна, а ключ відсутній, він додається.

Експерименти показують, що алгоритм добре працює на початку заповнення таблиці, проте у міру заповнення процес сповільнюється, а довгі серії проб стають все більш частими.

Квадратична і довільна адресація. Замість постійної зміни на одиницю, як у випадку з лінійною адресацією, можна скористатися наступною формулою [4]:

$$h = h + a^2,$$

де а – це номер спроби.

Цей вид адресації досить швидкий і передбачуваний. Чим більше колізій в таблиці, тим довше цей шлях.

Довільна адресація використовує заздалегідь згенерований список випадкових чисел для отримання послідовності.

Адресація з подвійним хешуванням. Цей алгоритм перевіряє таблицю трохи інакше, тримаючи її обома хеш-функції $h_1(K)$ і $h_2(K)$. Остання повинна породжувати значення в інтервалі від 1 до М – 1, взаємно прості з М.

Видалення елементів хеш-таблиці. Багато програмістів настільки сліпо вірять в алгоритми, що навіть не намагаються замислюватися над тим, як вони працюють. Для них неприємним сюрпризом стає те, що очевидний спосіб видалення записів з хеш-таблиці не працює. Взагалі кажучи, обробляти видалення можна, позначаючи елемент як видалений, а не як порожній. Таким чином, кожна клітинка в таблиці буде містити вже одне з трьох значень: порожня, зайнята, видалена. При пошуку вилучені елементи будуть трактуватися як зайняті, а при вставці – як порожні, відповідно.

Однак, очевидно, що такий метод працює тільки при рідкісних видаленнях, оскільки одного разу зайнята позиція більше ніколи не зможе стати вільною, а, значить, після довгої послідовності вставок і вилучень всі вільні позиції зникнуть, а при невдалому пошуку буде вимагатися М перевірок (де М, нагадаємо, розмір хеш-таблиці).

Розглянемо алгоритм видалення елемента і при лінійній адресації.

1. Позначити TABLE [i] як вільну позицію і встановити j = i.

2. i = i – 1 або i = i + M, якщо і стало негативним.

3. Якщо TABLE [i] порожній, алгоритм завершується, тому що немає більше елементів, про доступ до яких слід піклуватися. В іншому випадку ми встановлюємо r = h (KEY [i]), де KEY [i] – ключ, який зберігається в TABLE[i]. Це нам дасть його первісна хеш-адреса. Якщо i≤r<j або r<j<i або j<i≤ r (іншими словами, якщо r циклічно лежить між цими двома змінним, що свідчить про те, що цей елемент знаходиться в ланцюжку, ланку якої ми видалили вище), повернутися на крок 1.

4. Треба перемістити запис TABLE [j] = TABLE [i] і повернутися на перший крок.

Можна показати, що цей алгоритм не викликає зниження продуктивності. Однак, коректність алгоритму сильно залежить від того факту, що використовується лінійне дослідження хеш-таблиці, тому аналогічний алгоритм для подвійного хешування не існує.

Даний алгоритм дозволяє переміщати деякі елементи таблиці, що може виявитися небажано

(наприклад, якщо є посилання ззовні на елементи хеш-таблиці). Інший підхід до проблеми видалення ґрунтується на адаптуванні деяких ідей, що використовуються при складанні сміття: можна зберігати кількість посилань з кожним ключем, що говорить про те, як багато інших ключів стикається з ним. Тоді при обнуленні лічильника можна перетворювати такі осередки в порожні.

Застосування хешування. Одне з побічних застосувань хешування полягає в тому, що воно створює свого роду зліпок, «відбиток пальця» для повідомлення, текстового рядка, області пам'яті і т. п. Такий «відбиток пальця» може прагнути як до «унікальності», так і до «схожості». В цій якості однією з найважливіших областей застосування є криптографія. Тут вимоги до хеш-функцій мають свої особливості. Крім швидкості обчислення хеш-функції потрібно значно ускладнити відновлення повідомлення (ключа) за хеш-адресою. Необхідно відповідно ускладнити пошук іншого повідомлення з тією ж хеш-адресою. При побудові хеш-функції односпрямованого характеру зазвичай використовують функцію стиснення (видає значення довжини n при вхідних даних більше довжини m і працює з декількома вхідними блоками). При хешуванні враховується довжина повідомлення, щоб виключити проблему появи однакових хеш-адрес для повідомлень різної довжини. Найбільшу популярність мають такі хеш-функції: MD4, MD5, RIPEMD-128 (128 біт), RIPEMD-160, SHA (160 біт).

Хешування паролів. Нижче передбачається, що для шифрування використовується 128-бітний ключ. Зрозуміло, це не більше, ніж конкретний приклад. Хешування паролів – метод, що дозволяє користувачам запам'ятовуються не 128 байт, тобто 256 шістнадцяткових цифр ключа, а деякий осмислений вираз, слово або послідовність символів, що називається паролем. Дійсно, при розробці будь-якого криптоалгоритму слід враховувати, що в половині випадків кінцевим користувачем системи є людина, а не автоматична система. Це ставить питання про те, чи зручно, і взагалі чи реально людині запам'ятати 128-бітний ключ (32 шістнадцяткові цифри). Насправді межа запам'ятовування лежить на кордоні 8-12 подібних символів, а, отже, якщо ми будемо змушувати користувача оперувати саме ключем, тим самим ми практично змусимо його до запису ключа на якомусь клаптику паперу або електронному носії, наприклад, в текстовому файлі. Це, природно, різко знижує захищеність системи.

Для вирішення цієї проблеми були розроблені методи, що перетворюють промовлений, осмислений рядок довільної довжини – пароль, в зазначений ключ заздалегідь заданої довжини. У переважній більшості випадків для цієї операції використовуються так звані хеш-функції. Хеш-функцією в даному випадку називається таке математичне або алгоритмічне перетворення заданого блоку даних, яке має такі властивості:

1. Хеш-функція має нескінченну область визначення.
2. Хеш-функція має кінцеву область значень.
3. Вона необоротна.
4. Зміна вхідного потоку інформації на один біт змінює близько половини всіх біт вихідного потоку, тобто результату хеш-функції.

Ці властивості дозволяють подавати на вхід хеш-функції паролі, тобто текстові рядки довільної довжини будь-якою національною мовою і, обмеживши область значень діапазоном $0..2^N - 1$, де N – довжина ключа в бітах, отримувати на виході досить рівномірно розподілені по області значення блоки інформації – ключі.

Висновки з даного дослідження. В цій роботі розглянуто найбільш відомі алгоритми хешування, визначені їх недоліки та переваги, а також представлена реалізація обраного в результаті аналізу алгоритмів, так званого «оптимального» алгоритму для конкретної практичної задачі, яку перед нами поставив замовник.

Список літератури:

1. Шнайер, Брюс. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Брюс Шнайер. – М.: Триумф, 2002. – ISBN 5-89392-055-4.
2. Кнут Дональд. Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming, vol. 3. Sorting and Searching / Дональд Кнут. – 2-е издание. – М.: Вильямс, 2007. – С. 824. – ISBN 0-201-89685-0.
3. Кормен Т. Алгоритмы: построение и анализ. / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: МЦНМО, 2001.
4. Вирт Никлаус. Алгоритмы и структуры данных / Никлаус Вирт. – М.: Мир, 1989. – ISBN 5-03-001045-9.

Борд Р.В.
Днепропетровский национальный университет имени Олеся Гончара
Лозовська Л.И.
Национальная металлургическая академия Украины

МЕТОДОЛОГИЧЕСКИЙ ВЫБОР ХЕШ-ФУНКЦИИ ДЛЯ ОПТИМАЛЬНОЙ СИНХРОНЗАЦИИ БАЗЫ ДАННЫХ НАЦИОНАЛЬНЫХ ПАРКОВ ХОРВАТИИ

Аннотация
К основным вопросам, возникающим во время поиска оптимального алгоритму хеширования, относится исследование таких свойств этих алгоритмов как устойчивость к поиску первого та второго прообразов, вычислительная сложность. В работе проведен анализ наиболее распространенных алгоритмов построения хеш-функций. Исследованы их достоинства и недостатки. Определены основные критерии подбора оптимальной хеш-функции для конкретных практических задач. Определен оптимальный алгоритм хеширования для конкретной практической задачи.
Ключевые слова: алгоритм хеширования, MD4, MD5, SHA, динамическое хеширование, метод умножения, метод раздела, минимальное идеальное хеширование.

Bord R.V.
Dnipropetrovsk Oles Honchar National University
Lozovskaya L.I.
National Metallurgical Academy of Ukraine

METHODOLOGICAL CHOICE OF HASH FUNCTIONS FOR OPTIMAL SYNCHRONIZATION OF THE DATABASE OF NATIONAL PARKS IN CROATIA

Summary
The main issues that arise during the optimal hashing algorithm searching are the studies of these algorithms properties such as resistance to find the first and second prototypes, computational complexity. The article discovers analysis of the most common hash functions algorithms. There are also covered its advantages and disadvantages. We clarified the main criteria for optimal hash function selection within specific practical problems. The optimal hashing algorithm is identified for a certain customer's practical problem.
Keywords: hashing algorithm, MD4, MD5, SHA, dynamic hashing method of multiplication, division method, minimal perfect hashing.