Борд Р.В.
студент,
Дніпропетровський національний університет
імені Олеся Гончара
Науковий керівник: **Лозовська Л.І.**
кандидат фізико-математичних наук, доцент,
Національна металургійна академія України

АНАЛІЗ ВАРІАНТІВ ВИБОРУ ХЕШ-ФУНКЦІЇ ДЛЯ ОПТИМАЛЬНОЇ СИНХРОНІЗАЦІЇ БАЗИ ДАНИХ

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, 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 властивостей цих алгоритмів. Зокрема, для розв'язання задачі оптимізації синхронізації бази даних національних парків Хорватії потрібно було проаналізувати доступні алгоритми хешування та знайти такий алгоритм, який б водночас був і стійким до пошуку прообразів, але дуже швидко міг видавати результат хешування для дуже великих об'ємів вхідних даних.

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.

Метою даної роботи є аналіз основних критеріїв підбору оптимальної хеш-функції для конкретних практичних задач, а також визначення оптимального алгоритму хешування для конкретної практичної задачі.

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. Якісна хеш-функція повинна задовольняти двом вимогам: обчислення повинно виконуватися дуже швидко; вона повинна мінімізувати кількість колізій. Теоретично неможливо визначити хеш-функцію так, щоб вона створювала випадкові дані з реальних невипадкових файлів. Однак на практиці реально створити досить хорошу імітацію. Більш

того, часто використовуються особливості даних для створення хеш-функцій з мінімальною кількістю колізій [3].

Виокремлюють наступні методи хешування:

Метод поділу [2]: h (K) = K mod M.

Потребує ретельно вибору константи M (доцільно використовувати просте число). У більшості випадків подібний вибір цілком задовільний.

Метод множення [2]: h(K) = [M * ((C * K) mod 1)].

Тут, якщо константа C обрана вірно, то можна домогтися дуже хороших результатів, однак, цей вибір складно зробити. Мультиплікативний метод користується тим, що реальні файли невипадкові. Наприклад, часто множиною ключів є арифметична прогресія ключів {K, K+d, K+2d,..., K+td}.

Окремим випадком вибору константи є значення величини золотого перетину φ. 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 [2] і доведено С. Сверчковскі [2]. Стосовно до хешування це означає, що якщо в якості константи вибрати золотий перетин, то функція буде досить добре розсіювати ключі.

Алгоритм MD4. Виокремлюють основні цілі алгоритму MD4 [3]:

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

2. *Швидкість:* програмна реалізація алгоритму повинна виконуватися досить швидко (алгоритм повинен бути швидким для 32-бітної архітектури).

3. *Простота та компактність:* алгоритм повинен бути простим в описі і в програмуванні, без великих програм або підстановочних таблиць.

4. *Бажано little-endian архітектура:* деякі архітектури процесорів зберігають ліві байти слова в позиції молодших адрес байта. Інші зберігають праві байти слова в позиції молодших адрес байта. Ця відмінність важлива, коли повідомлення трактується як послідовність 32-бітових слів, і враховувалася при розробці алгоритму MD5 [3], який є більш складним і більш повільним ніж MD4. Але додавання складності виправдовується зростанням рівня безпеки.

Динамічне хешування дозволяє динамічно змінювати розмір хеш-структури. Хеш-функція генерує псевдоключ, який використовується лише частково для доступу до елементу (генерується бітова послідовність, яка повинна бути достатньою для адресації всіх потенційно можливих елементів).

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 використовує список, елементи якого посилаються на блоки. Самі елементи адресуються за деякою кількістю *i* бітів псевдоключа. При пошуку береться *i* бітів псевдоключа і через список знаходиться адреса блоку. Для додавання елементів спочатку виконується процедура 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. Можлива ситуація, коли кілька елементів показують

на один і той же блок. 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.

Крім вибору хеш-функції, необхідно реалізувати механізм *розв'язання колізій*. 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, 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 K 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 K. Якщо при пошуку зустрічається порожня клітинка, то можна зробити висновок, що K в таблиці відсутнє.

Лінійна адресація використовує циклічну послідовність перевірок [1]:

$$h(K), h(K-1),..., 0, M-1, M-2,..., h(K)+1.$$

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

Квадратична і довільна адресація. Замість постійної зміни на одиницю, можна скористатися наступною формулою [4]:

$$h = h + a^2, \text{ (a – номер спроби).}$$

Цей вид адресації досить швидкий і передбачуваний, але чим більше колізій в таблиці, тим довший цей шлях. Довільна адресація використовує заздалегідь згенерований список випадкових чисел для отримання послідовності.

Адресація з подвійним хешування. 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 M – 1, взаємно прості з M.

Видалення елементів хеш-таблиці. Обробляти видалення можна, позначаючи елемент як видалений, а не порожній. Таким чином, кожна клітинка таблиці буде містити одне з трьох значень: порожня, зайнята, видалена. При пошуку вилучені елементи будуть трактуватися як зайняті, а при вставці – порожні.

Розглянемо алгоритм видалення елемента *i* при лінійній адресації.

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

2. $i = i - 1$ або $i = i + M$, якщо *i* стало негативним.

3. Якщо *TABLE [i]* порожня, алгоритм завершується, тому що немає більше елементів, про доступ до яких слід піклуватися. Інакше ми встановлюємо *r=h (KEY [i])*, де *KEY [i]* – ключ, який зберігається в *TABLE [i]*. Якщо $i \le r < j$ або $r < j < i$ або $j < i \le r$ (іншими словами, якщо *r* циклічно лежить між цими двома змінним, що свідчить про те, що цей елемент знаходиться в ланцюжку, ланку якого ми видалили вище), повернутися на крок 1.

4. Перемістити запис TABLE [j]=TABLE [i] і повернутися на крок 1.

Можна показати, що цей алгоритм не викликає зниження продуктивності. Однак, коректність алгоритму сильно залежить від того факту, що використовується лінійне дослідження хеш-таблиці. Даний алгоритм дозволяє переміщати деякі елементи таблиці, що може виявитися небажано. 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.

Список використаних джерел:

1. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер. – М.: Триумф, 2002.

2. Кнут Д. Искусство программирования. Том 3. Сортировка и поиск = The Artof Computer Programming, vol. 3. Sorting and Searching / Дональд Кнут. – 2-е издание. – М.: Вильямс, 2007. – С. 824.

3. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: МЦНМО, 2001.

4. Вирт Н. Алгоритмы и структуры данных / Н. Вирт. – М.: Мир, 1989.

Бронза С.Д.

кандидат фізико-математичних наук, доцент,
Харківський державний університет залізничного транспорту

Васильченко Н.В.

асистент,
Краснолиманська філія
Харківського університету залізничного транспорту

ЗВІЛЬНЕННЯ ВІД ІРРАЦІОНАЛЬНОСТІ

Важливе місце в курсі алгебри посідають симетричні многочлени та, зокрема, застосування симетричних многочленів при розв’язуванні рівнянь, систем рівнянь, вилучення коренів, доведення тотожностей, звільнення від ірраціональності у дробах тощо. Цими питаннями займалися багато вчених, зокрема, Франсуа Вієт.

Метод симетричних многочленів відноситься до числа поліноміальних, але принципово відрізняється від вищезгаданих тим, що для обчислення функцій матриці не вимагає знаходження рішень характерного рівняння, а базується лише на використанні коефіцієнтів цього рівняння.

Франсуа Вієт розробив ряд важливих питань теорії рівнянь 1-4 степенів. Він сформулював і довів кілька теорем про взаємозв’язки між коренями і коефіцієнтами рівнянь, зокрема, й теорему про зведене квадратне рівняння (теорема Вієта). На сьогоднішній день теорема Вієта є необхідною і важливою частиною шкільної програми [4, с. 34].