МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н. Каразіна

Навчально-науковий інститут комп’ютерних наук та штучного інтелекту

Курсова робота

з дисципліни «Теорія алгоритмів»

Тема: «Хешування»

Виконав: студент 2 курсу групи КС22
Спеціальності 122 «Комп’ютерні науки»
Скрипняк Тарас Артемович
Прийняв: Олешко О.І.

Харків - 2024

ЗМІСТ

ВСТУП

Хешування є одним з основних методів захисту інформації в інформаційних технологіях, широко застосовуваним у криптографії та базах даних для забезпечення безпеки та цілісності даних. У криптографії вони забезпечують цілісність і захист інформації, що використовується для автентифікації, цифрових підписів і зберігання паролів. Але крім цього, хеш-функції мають широке застосування в оптимізації, зберігання цілосності та достовірності даних, тощо.

Зокрема, у базах даних хешування використовується для прискорення пошуку даних і забезпечення ефективного доступу до них. У комп’ютерних мережах воно служить для створення контрольних сум, що допомагає виявляти та виправляти помилки при передачі інформації. Хеш-функції також активно застосовуються в блокчейн-технологіях для створення захищених реєстрів транзакцій та забезпечення незмінності даних.

Завдяки своїм унікальним властивостям — швидкості, ефективності та здатності створювати унікальні ідентифікатори для будь-яких даних — хеш-функції є незамінними інструментами, які продовжують знаходити нові застосування у сферах, пов’язаних із безпекою, обробкою великих даних та оптимізацією процесів.

Ця курсова робота присвячена аналізу основних аспектів хешування: принципам побудови хеш-функцій, вимогам до їх властивостей та прикладам застосування в різних сферах.

РОЗДІЛ 1. ПРИНЦИПИ ТА ОСНОВИ ХЕШУВАННЯ

1.1 Принцип, історія та загальний огляд деяких хеш-функцій

Хеш-функція — це математична функція, яка перетворює строку вхідних даних у фіксований розмір вихідного рядка, який називають хешем або хеш-кодом. Цей процесс перетворення будь-яких вхідних даних у хеш називається хешуванням. Хеш-функції мають декілька важливих властивостей:

1. Детермінованість — однакове вхідне значення завжди має створювати однаковий хеш або алгоритм порівняння різних хешів. Це дозволяє відновлювати хеш із тих самих даних для порівняння.
2. Швидкість обчислення — хеш-функція має швидко обробляти дані, навіть якщо їхній обсяг великий. Це важливо для ефективної роботи з великими наборами даних. У цього є вийняткі, такі, як хеш-функція для паролів bcrypt, що за ціну швидкості гарантує стійкість проти brute-force атак.
3. Непридатність до інверсії — для криптографічних хеш-функцій зворотне перетворення хешу у вихідні дані має бути обчислювально неможливим. Це особливо важливо для захисту вихідних даних (наприклад, паролі).
4. Стійкість до колізій — бажано, щоб було надзвичайно складно знайти два різні вхідні значення, які б давали однаковий хеш. Це гарантує унікальність хешу для кожного унікального набору вхідних даних.
5. Avalanche-ефект — невелика зміна у вхідних даних має значно змінити хеш. Це дозволяє виявляти навіть незначні зміни у вихідних даних через порівняння хеш-кодів.

Перші концепції, які згодом стали основою для хеш-функцій, виникли у сфері пошуку та сортування даних. У цей період розробники баз даних та комп'ютерних систем шукали способи оптимізації зберігання інформації та швидкого доступу до неї. Ідея хешування полягала у створенні унікальних

ідентифікаторів, які дозволяли уникати дублювання та пришвидшувати процеси обробки даних.

Перша систематична ідея хешування приписується співробітнику IBM Хансу Петеру Луну, який запропонував алгоритм хеш-кодування (Key Word in Context) у січні 1953 року. Це можна вважати початком концепції хешування в сучасному розумінні. У 1956 році Арнольд Думи зробив значний внесок у розвиток хешування роботою під назвою “Computers and automation”, де представив концепцію хешування у формі, близькій до сучасного розуміння програмістів[1].

Подальша загальна історія хешування була незначною, але впродовж наступних десятиліть розробники зосередилися на створенні все більш надійних і швидкодійних алгоритмів, здатних ефективно обробляти великі обсяги даних та протистояти спробам розкриття. Це призвело до створення широковідомих алгоритмів, таких як Message Digest в 1989р та Secure Hash Algorithm в 1993р, які стали стандартами в хешуванні.

Message Digest — одна з найпопулярніших (але в основному, нажаль, застарілих) криптографічних хеш-функцій. Розроблена Робертом Рівестом, друга версія алгоритму була опублікована в 1986р компанією RSA Security як частина криптосистеми RSA (перша версія була пропрієтарною). MD2, як вона була названа, реалізувала алгоритм перетворення вхідних даних у 128 бітний односторонній хеш[2]. Більшість версій Message Digest — MD2, MD4 та MD5 — на сьогоднішній день, нажаль, скомпрометовані і не рекомендуються для використання як надійні односторонні хеш-функції. MD6, остання версія алгоритму, була опублікована в 2008р. та на сьогоднішній день остається однією з безпечних криптографічних хеш-функцій. На сьогоднішній день MD5 досить часто використовується для чек-сум. [3]

Secure Hash Algorithms — це родина алгоритмів хешування, які в основному розроблюються за ініціативою уряду Сполучених Штатів Америки.

Перша версія, SHA-0 або SHA-1, була розроблена Агенством національної безпеки в 1993р в рамкак проекту Capstone, та взагалом використовувала принципи хешування схожі на методи Роберта Рівеста для алгоритму Message Digest. SHA-1 продукує хеш розміром в 160 бітів. На сьогоднішній день SHA-1 є скомпрометованою та не рекомендується експертами для захисту даних, але алгоритм досить використовується для перевірки цілосності даних у системах контролю версій, таких, як Git, Mercurial та Monotone[4].

SHA-2 — це коллекція хеш-функцій другої версії Secure Hash Algorithm, а точніше SHA-224/256/384/512, кожна з яких продукує хеш розміру відповідного назві — 224/256/384/512 бітів. На даний момент SHA-2 найвикористовуємий криптологічний алгоритм хешування, т.я. його практично не можливо реконструювати. Він використовується повсюди — від безпечних веб-протоколів як TLS/SSL, до аутентифікації програмного забезпечення на дистрибутивах системи Linux, а також родина SHA використовується у системах Агенції національної безпеки Сполучених Штатів, для захисту засекреченої інформації[5].

1.2 Хеш-таблиці, іх теоритичний аналіз та хеш-функції для створення індексів

1.2.1 Що таке хеш-таблиця.

Хеш-таблиця — це структура даних, яка дозволяє ефективно зберігати та шукати дані за ключем. Вона працює на основі хешування, де спеціальна хеш-функція перетворює ключ (наприклад, ім'я користувача або номер) у певне числове значення — хеш-код. Цей хеш-код використовується як індекс для зберігання значення в масиві, що дозволяє швидко знайти потрібний елемент[6]. (див. рис. 1.1)

Принцип роботи хеш-таблиць

1. Хешування ключа: Коли новий елемент додається до хеш-таблиці, спочатку до його ключа застосовується хеш-функція, яка перетворює ключ у числовий індекс.
2. Зберігання значення: На отриманому індексі в масиві зберігається значення, пов'язане з ключем. Це значно пришвидшує доступ до даних, оскільки звернення до значення відбувається за відомим індексом масиву.
3. Пошук значення: Щоб знайти значення за ключем, алгоритм застосовує ту саму хеш-функцію до ключа, отримуючи індекс, на якому зберігається потрібне значення.

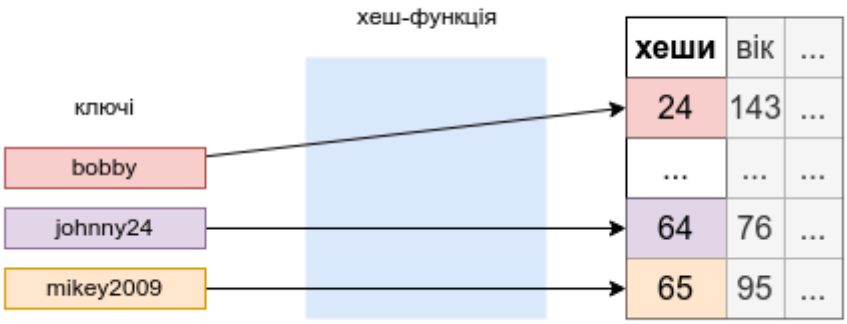

Рисунок 1.1 — Абстрактний опис принципу роботи хеш-таблиці

1.2.2 Хеш-функції для хеш-таблиць.

Для хеш-таблиць зазвичай використовуються спеціальні хеш-функції, які оптимізовані для швидкого обчислення і мінімізації колізій. Основними критеріями для вибору хеш-функції є її ефективність (швидкість обчислення) та рівномірний розподіл результатів. Деякі хеш-функції для хеш-таблиць:

1. Метод ділення
   - Проста і швидка функція, де ключ перетворюється в числовий формат, а потім береться залишок від ділення на розмір таблиці: $h(key) = key \pmod{size}$

2. Метод множення
    - Використовує множення ключа на константу, щоб отримати дробову частину, яка потім масштабується на розмір таблиці: $h(key) = \lfloor size(key * A \pmod 1) \rfloor$
    - Константа A зазвичай обирається як ірраціональне число (наприклад, золотий перетин $\frac{\sqrt{5}-1}{2}$, що зменшує шанси колізій[7].

1.3 Колізії та методи іх вирішення

1.3.1 Що таке колізії

Колізії в хеш-функціях — це ситуації, коли дві різні вхідні значення (ключі) мають однаковий хеш-код. У результаті вони зберігаються в одній комірці хеш-таблиці, що призводить до конфлікту, оскільки хеш-таблиця очікує, що кожне значення буде мати унікальний індекс.

Колізії є неминучими у хеш-таблицях через обмежений розмір хеш-простору. Хеш-функція перетворює будь-яке вхідне значення в хеш фіксованої довжини. Отже, коли різних вхідних значень більше, ніж можливих хешів, деякі з них матимуть однакові хеш-коди.

Приклади колізій:

1. Якщо використовується проста функція залишку (наприклад, $h(key) = key \pmod{10}$) для ключів 12 і 22, результатом буде однаковий хеш-код: 2. Це створює колізію.
2. У криптографічному хешуванні колізія також може виникнути, якщо дві різні рядкові фрази дають однаковий хеш-код. Такі колізії є вразливістю з точки зору безпеки, оскільки можуть дозволити зловмисникам обійти перевірку цілісності даних [8].

1.3.2 Методи розв'язання колізій

Для ефективного використання хеш-таблиць важливо мати стратегії для вирішення колізій:

1. Ланцюжки (окреме збереження) — кожна комірка таблиці містить список значень, які мапуються на цей індекс. Якщо виникає колізія, новий елемент додається в список, а при пошуку алгоритм перевіряє весь список у цій комірці.

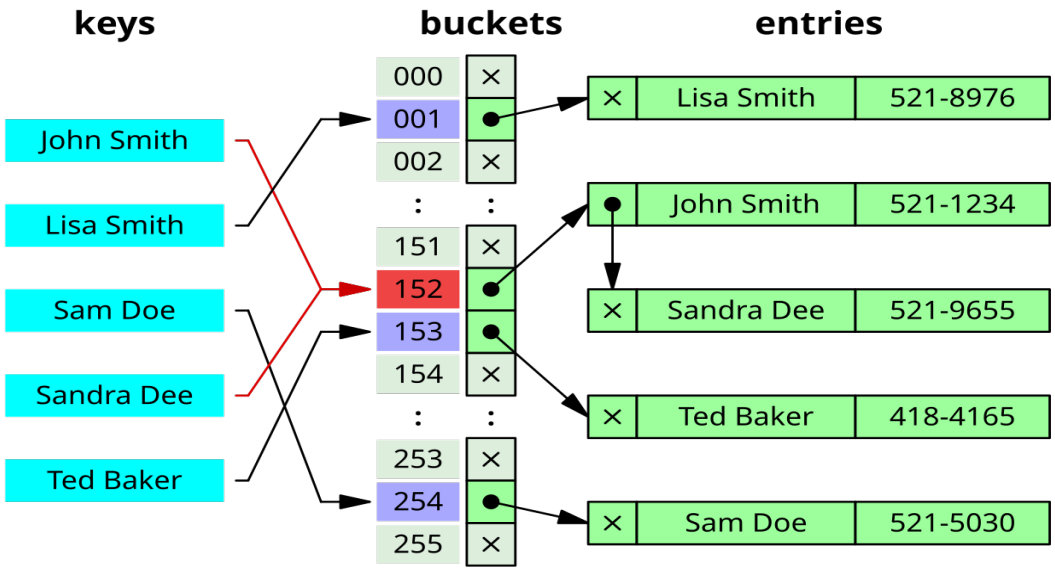

Рисунок 1.2 — Абстрактний опис алгоритму ланцюжків

2. Відкрита адресація — якщо комірка, в яку потрібно додати елемент, зайнята, алгоритм шукає іншу комірку за певним правилом:
   - Лінійне пробування — шукає наступну вільну комірку послідовно (наприклад, +1, +2…). (див. рис. 1.4)
   - Квадратичне пробування — шукає комірки зі зміщенням, яке збільшується квадратично (наприклад, +1, +4, +9…).
   - Подвійне хешування — застосовує додаткову хеш-функцію, щоб обчислити зміщення, що зменшує ймовірність збирання колізій у групи.

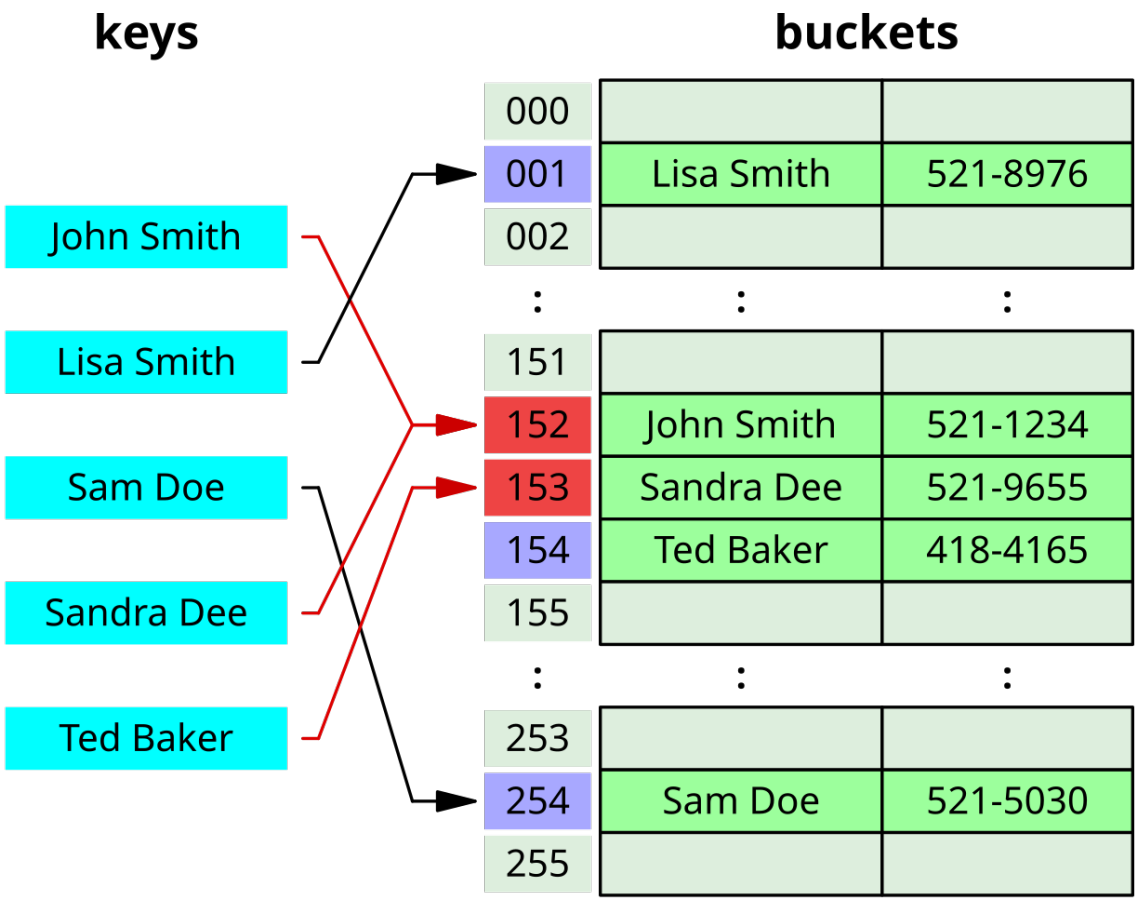

Рисунок 1.4 — Приклад рішення колізії за допомогою лінійного пробування

3. Перехешування — якщо кількість колізій у таблиці стає занадто великою, таблиця збільшується і всі елементи перехешовуються з новим розміром. Це дорогий, але ефективний метод для великих наборів даних[8].

4. Зозулине хешування — метод розв'язання колізій у хеш-таблицях, який забезпечує постійний час доступу O(1) навіть у найгіршому випадку. Принцип Cuckoo Hashing заснований на ідеї використання двох або більше хеш-функцій та переміщенні (витісненні) значень у таблиці, якщо виникає колізія.

    - Замість використання лише однієї хеш-функції, у Cuckoo Hashing використовується, як правило, дві (або більше) хеш-функції h1 і h2,

кожна з яких може мапувати ключ на інше місце в таблиці. Це дозволяє кожному елементу мати кілька потенційних позицій у таблиці.

- Витіснення елементів: При додаванні нового елемента до таблиці, спочатку перевіряється перша позиція h1(key). Якщо ця позиція зайнята, то елемент, який вже знаходиться у цій позиції, витісняється (подібно до поведінки зозулі, яка підкидає яйця у гнізда інших птахів). Витіснений елемент переміщається у свою альтернативну позицію h2 (key). Якщо ця позиція також зайнята, то інший елемент знову витісняється та переміщується, і процес триває, поки не буде знайдено вільне місце, або не досягнеться гранична кількість спроб.
- Рехешування у випадку зациклення: Якщо елемент не може знайти вільне місце протягом певної кількості спроб (зазвичай 10–20), тоді таблиця "перехешовується": її розмір збільшується, і всі елементи перехешовуються, використовуючи нові хеш-функції[9].

1.4 Порівняльний аналіз хеш-функцій та методів вирішення колізій

1.4.1 Порівняльний аналіз хеш-функцій

Аналіз різних хеш-функцій зосереджується на їхніх основних характеристиках, таких як ефективність обчислення, рівномірність розподілу, стійкість до колізій і випадки використання.

Таблиця 1.1 — Порівняльний аналіз

Хеш-функція	Переваги	Недоліки	Застосування
Метод ділення	Дуже проста й швидка у виконанні; підходить для числових ключів.	Погано розподіляє ключі з певними закономірностями (наприклад, послідовні значення); чутлива до розміру таблиці.	Невеликі таблиці, де потрібна максимальна швидкість, і ключі не мають послідовних закономірностей.

Метод множення	Добре розподіляє ключі; забезпечує рівномірний розподіл за правильно підібраних параметрів.	Необхідно обрати підходящу константу A; складніша реалізація, ніж модульне хешування.	Загальні випадки, де потрібен рівномірний розподіл для числових або коротких ключів.
Криптографічні хеш-функції (SHA, MD5)	Дуже висока стійкість до колізій, чудова безпека для критичних даних; забезпечує "стійкість" хешів до змін.	Повільні через складність обчислень; надлишкова для звичайних хеш-таблиць, де не потрібна криптографічна безпека.	Використовуються для безпеки, перевірки цілісності, але не для звичайних хеш-таблиць; більше підходять для хешів паролів та підписів.

1.4.2. Порівняльний аналіз методів рішення колізій

1. Ланцюжки (Chaining)

   Суть методу: Кожен елемент у хеш-таблиці є списком або зв’язаним списком. Коли кілька елементів мають один і той самий хеш-індекс, вони зберігаються у цьому списку.

   - Переваги:
     - Легко реалізується.
     - Не потребує додаткового місця у хеш-таблиці.
     - Просте масштабування: коли кількість колізій зростає, списки стають довшими, а не займають більше місця в таблиці.
   - Недоліки:
     - Витрати на пам'ять для кожного списку.
     - Швидкість доступу знижується у випадку великої кількості колізій.

- Складність:
  - В середньому: $O(1)$
  - У гіршому випадку (якщо всі елементи потрапляють в один список): $O(n)$

2. Лінійне пробування (Linear Probing)

   Суть методу: Якщо місце для елемента зайняте, він розміщується в наступній доступній комірці (додаючи 1, 2 тощо до хеш-індексу).

   - Переваги:
     - Проста реалізація та зручне використання пам’яті.
     - Підтримує ефективну вставку, пошук і видалення за наявності низької заповнюваності.
   - Недоліки:
     - Проблема кластеризації: послідовні колізії накопичуються, збільшуючи час пошуку.
     - Час пошуку збільшується при високому коефіцієнті заповнення.
   - Складність:
     - В середньому: $O(1)$
     - У гіршому випадку: $O(n)$

3. Квадратичне пробування (Quadratic Probing)

   Суть методу: Подібно до лінійного пробування, але індекси шукаються не послідовно, а за квадратичною формулою (наприклад, $T[h(k)], T[h(k) + 1^2], T[h(k) + 2^2]...$)

   - Переваги:
     - Менше кластеризації, ніж у лінійному пробуванні.

- Легко реалізується та знижує навантаження на сусідні комірки.

- Недоліки:
  - Може знадобитися додаткова логіка для уникнення нескінченних циклів.
  - Потрібен специфічний розмір хеш-таблиці, наприклад, просте число, щоб запобігти колізіям.

- Складність:
  - В середньому: $O(1)$
  - У гіршому випадку: $O(n)$

4. Подвійне хешування (Double Hashing)

   Суть методу: Використовує два різні хеш-функції. Якщо перша викликає колізію, друга обчислює новий хеш-індекс для кожної спроби.

   - Переваги:
     - Зменшує кластеризацію та більш рівномірно розподіляє значення.
     - Підходить для великих хеш-таблиць, оскільки добре розподіляє колізії.
   - Недоліки:
     - Складніша у реалізації через потребу в двох різних хеш-функціях.
     - Виконує більше обчислень для кожної колізії.
   - Складність:
     - В середньому: $O(1)$
     - У гіршому випадку: $O(n)$, якщо обидві хеш-функції дають численні колізії.

5. Cuckoo Hashing

   Суть методу: Використовує дві хеш-таблиці і дві хеш-функції. Якщо обидві позиції зайняті, елемент "викидає" інший, який потім переміщується на інше місце за своєю другою хеш-функцією.

   - Переваги:
     - Гарантує $O(1)$ для пошуку.
     - Використовує менше пам’яті, оскільки немає необхідності в списках або пробуванні.
   - Недоліки:
     - При високому коефіцієнті заповнення виникає потреба у реорганізації таблиці (реструктуризації).
     - Складний у реалізації, особливо для великих таблиць.
   - Складність:
     - В середньому: $O(1)$
     - У гіршому випадку: $O(1)$

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Хеш-функція – uk.wikipedia.org
2. MD2 – en.wikipedia.org
3. MD5 – en.wikipedia.org
4. SHA-1 – en.wikipedia.org
5. SHA-2 – en.wikipedia.org
6. Basics of Hash Tables – hackerearch.com
7. Лекція 12. Хешування
8. Hash collision – en.wikipedia.org
9. Cuckoo sorting – levelup.gitconnected.com