Алгоритми хешування даних

Мета лекції

Розглянути:

- визначення та види хешування
- методи вирішення колізій в хеш таблицях
- основні алгоритми хешування

Загальні відомості

В області комп’ютерних наук та комп’ютерного програмування тип даних або просто тип є атрибутом даних, який повідомляє компілятору або інтерпретатору про те, як будуть використовувати дані.

Більшість мов програмування підтримують загальні типи даних числові, символьні та логічні.

Тип даних обмежує значення, які може мати вираз, наприклад, змінна або функція. Кожен тип даних визначає операції, які можуть бути зроблені на даних, значення даних, а також способи збереження значень цього типу.

Загальні відомості

Є багато класифікацій структур даних.

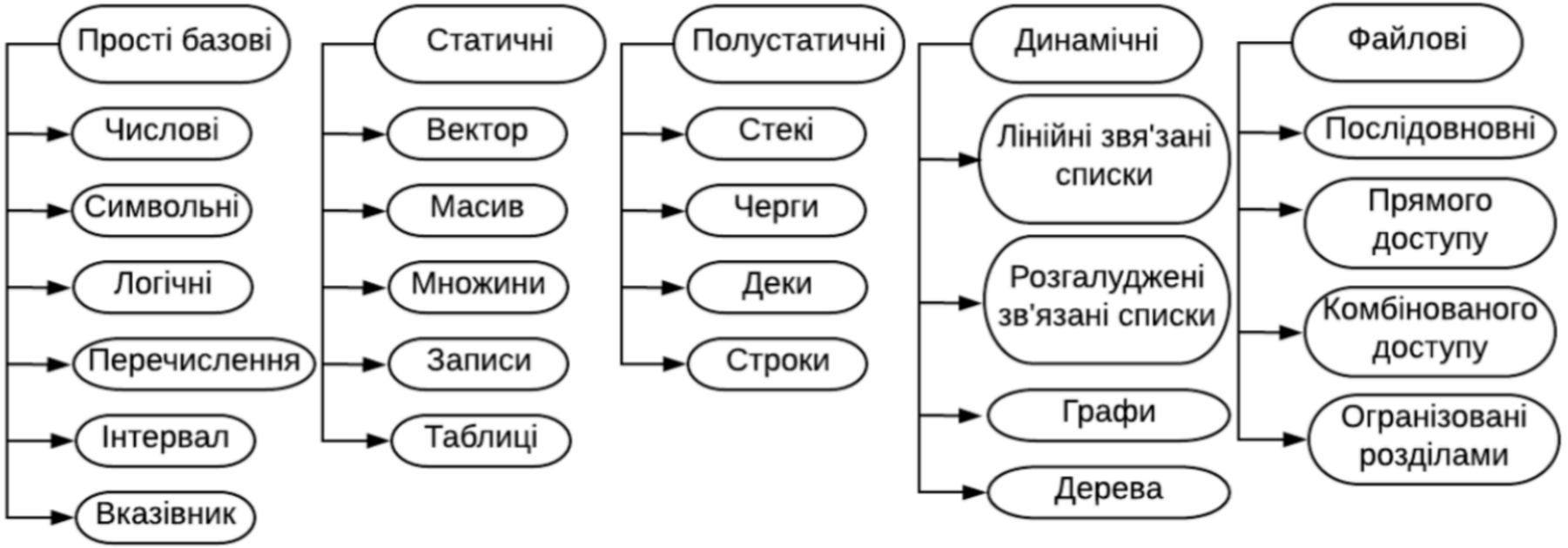


Рис 1. Структури даних

Загальні відомості

Також є варіант класифікації типів структур даних: на лінійну, ієрархічну і табличну.

Лінійна структура - це впорядкована множина у якій адрес елемента даних суворо визначається його впорядкованим номером. Прикладом може бути список.

Ієрархічна структура - це множина підмножин, що підмножини меншого ранга входять у підмножини вищого аж допоки не об’єднаються всі у єдину множину. До ієрархічної структури можна віднести дерева.

Таблична структура - це двовимірний масив, або матриця, що зберігає послідовність значень певного типу. При створенні масиву кожному елементу призначається номер та індекс, за яким його можна знайти у будь-який момент у масиві. Тобто на відміну від лінійної структури, у якій у кожного елемента є лише номер, у табличній структурі кожен елемент буде мати 2 номери - рядок та стовпець.

Загальні відомості

В програмуванні важливо не лише збирати і зберігати дані, а також мати доступ до них та можливість обробки. Дані можуть зберігатися у символьних таблицях, бінарних деревах, збалансованих деревах і хеш таблицях, що полегшує подальший пошук.

Основні поняття хешування даних

Процес пошуку даних у великих обсягах інформації пов’язаний з тимчасовими витратами, які обумовлені необхідністю перегляду та порівняння з ключем пошуку значного числа елементів.

Скорочення пошуку можливо здійснити шляхом локалізації області перегляду. Наприклад, впорядкувати дані по ключу пошуку, розбити на непересічні блоки по деякою груповою ознакою або поставити у відповідність реальним даним якийсь код, який спростить процедуру пошуку.

В даний час використовується широко поширений метод забезпечення швидкого доступу до інформації, що зберігається в зовнішній пам’яті - **хешування**.

Основні поняття хешування даних

Хешування (англ. Hashing) - це перетворення вхідного масиву даних певного типу та довільної довжини в вихідний бітовий рядок фіксованої довжини. Такі перетворення також називаються хеш-функціями або функціями згортки, а їх результати називають хешем, хеш-кодом, хеш-таблицею або дайджестом повідомлення (англ. Message digest).

Хеш-таблиця - це структура даних, що реалізує інтерфейс асоціативного масиву, тобто вона дозволяє зберігати пари виду «ключ-значення» і виконувати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення пари по ключу. Хеш-таблиця є масивом, яка формується в певному порядку хеш-функцією.

Асоціативний масив (англ. associative array або associative container, map, mapping, hash, dictionary, finite map) — абстрактний тип даних, що дозволяє зберігати дані у вигляді набору пар ключ — значення та доступом до значень за їх ключем .

Основні поняття хешування даних

Хеш-таблиця є ефективною структурою даних для реалізації словників.

Хоча на пошук елемента в хеш-таблиці може в найгіршому випадку знадобитися стільки ж часу, як і у зв’язаному списку, а саме **O(n)**, на практиці хешування дуже ефективне.

При досить обґрунтованих припущеннях математичне очікування часу пошуку елемента в хеш-таблиці складає **O(1)**.

Основні поняття хешування даних

Прийнято вважати, що хорошою, з точки зору практичного застосування, є така хеш-функція, яка задовольняє таким умовам:

• функція повинна бути простою з обчислювальної точки зору;

• функція повинна розподіляти ключі в хеш-таблиці найбільш рівномірно;

• функція не повинна відображати будь-який зв’язок між значеннями ключів в зв’язок між значеннями адрес;

• функція повинна мінімізувати число колізій - тобто ситуацій, коли різним ключам відповідає одне значення хеш-функції (ключі в цьому випадку називаються синонімами).

При цьому перша властивість хорошої хеш-функції залежить від характеристик комп’ютера, а друге - від значень даних.

Основні поняття хешування даних

Якби всі дані були випадковими, то хеш-функції були б дуже прості (наприклад, кілька бітів ключа). Однак на практиці випадкові дані зустрічаються досить рідко, і доводиться створювати функцію, яка залежала б від усього ключа.

Якщо хеш-функція розподіляє сукупність можливих ключів рівномірно по безлічі індексів, то хешування **ефективно розбиває** безліч ключів.

Найгірший випадок - коли всі ключі хешіруються в один індекс.

Основні поняття хешування даних

При виникненні колізій необхідно знайти нове місце для зберігання ключів, які претендують на одну і ту ж комірку хеш-таблиці.

Причому, якщо колізії допускаються, то їх кількість необхідно мінімізувати. У деяких спеціальних випадках вдається уникнути колізій взагалі. Наприклад, якщо всі ключі елементів відомі заздалегідь (або дуже рідко змінюються), то для них можна знайти деяку ін`єкційну хеш-функцію, яка розподілить їх по осередках хеш-таблиці без колізій.

Хеш-таблиці, що використовують подібні хеш-функції, не потребують механізму вирішення колізій, і називаються **хеш-таблицями з прямою адресацією**.

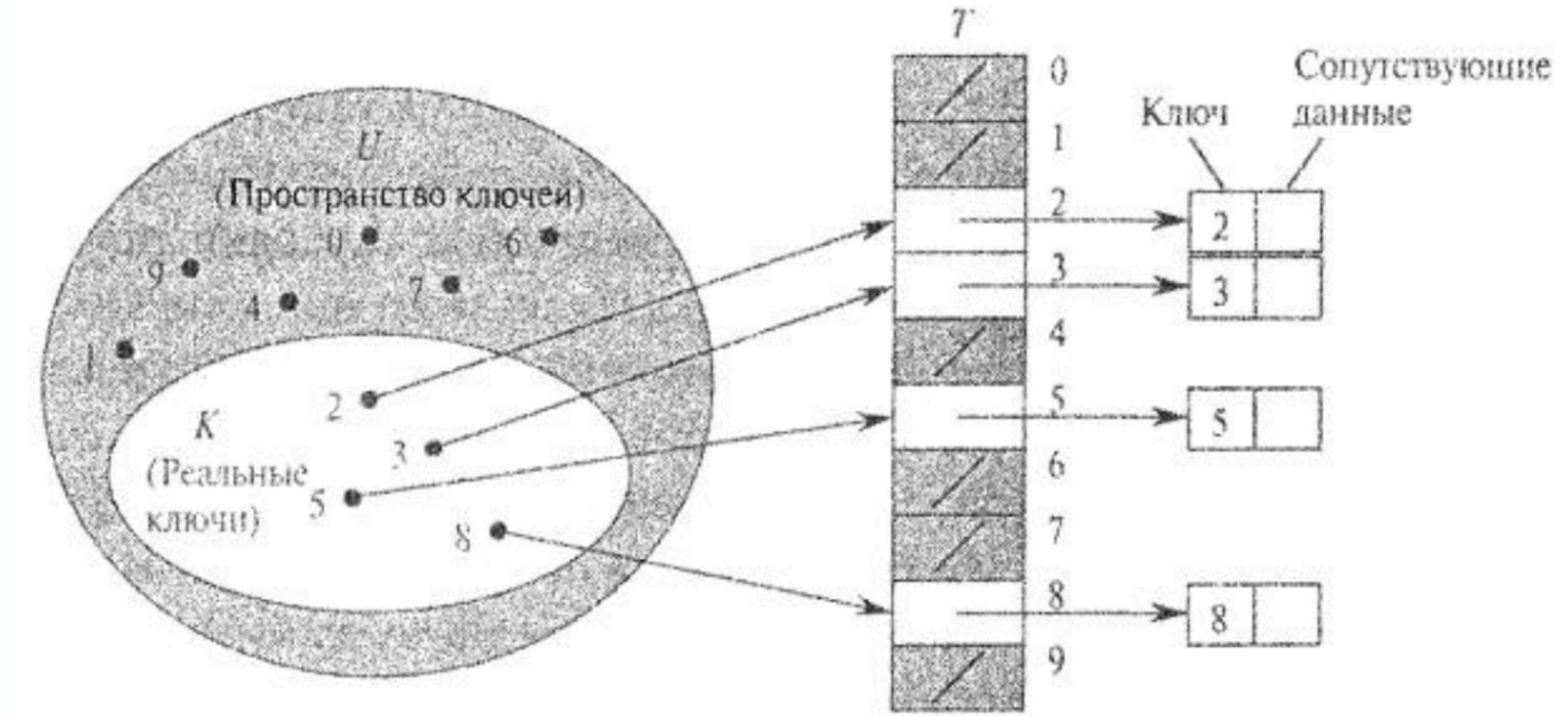
Основні поняття хешування даних

Пряма адресація є елементарною технологію, що добре працює для невеликих множин ключів. Нехай є певна множина, кожний елемент якої має ключ із множини U={0,1,..., m-1}, де m не дуже велике. Крім того, вважатимемо, що ніякі два елементи не мають однакових ключів.

Для представлення динамічних множин ми використовуємо масив, або таблицю з прямою адресацією, котрий позначимо як

T[0..m-1], кожна позиція, чи комірка, якого відповідає ключу із простору ключів U.

Основні поняття хешування даних (хеш таблиця з прямою адресацією)



Основні поняття хешування даних

Хеш-таблиці повинні відповідати наступним властивостям.

- Виконання операції в хеш-таблиці починається з обчислення хеш-функції від ключа. Одержуємоє хеш-значення є індексом у вихідному масиві.

- Кількість збережених елементів масиву, поділене на число можливих значень хеш-функції, називається коефіцієнтом заповнення хеш-таблиці (load factor) і є важливим параметром, від якого залежить середній час виконання операцій.

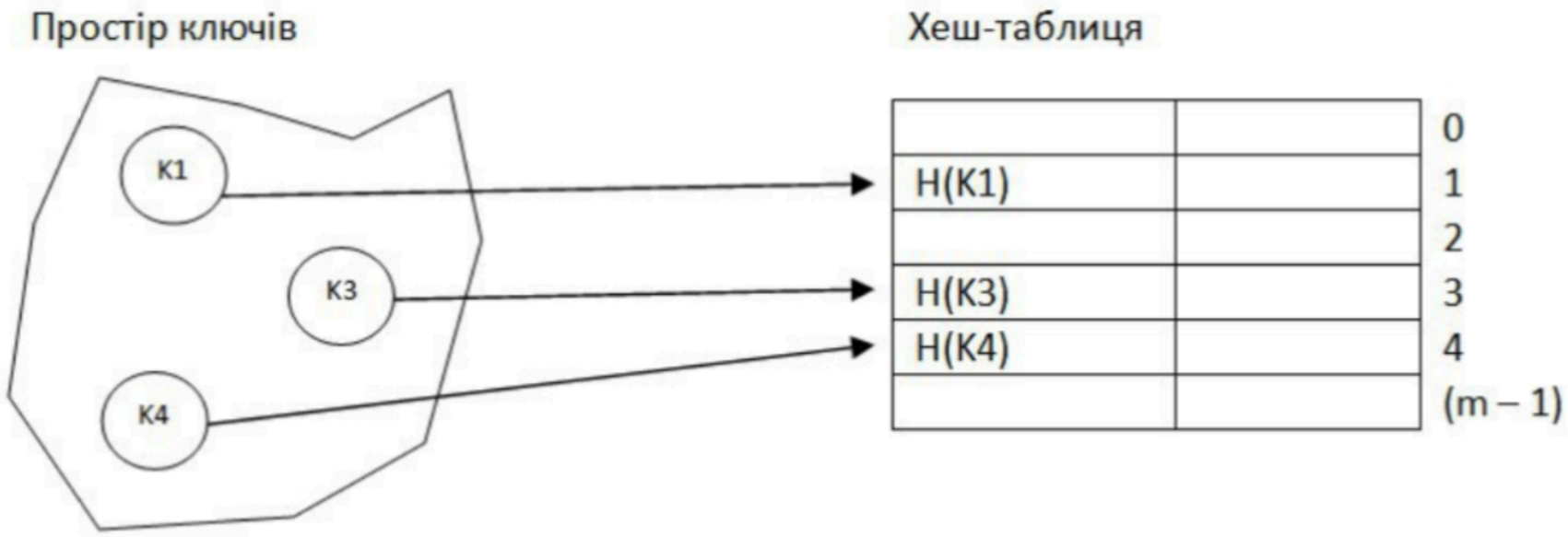
- Операції пошуку, вставки і видалення повинні виконуватися в середньому за час O(1). Однак при такій оцінці не враховуються можливі апаратні витрати на перебудову індексу хеш-таблиці, пов`язану зі збільшенням значення розміру масиву і додаванням в хеш-таблицю нової пари.

- Механізм вирішення колізій є важливою складовою будь-якої хеш-таблиці. Хешування корисно, коли широкий діапазон можливих значень повинен бути збережений в малому обсязі пам`яті, і потрібен спосіб швидкого, практично довільного доступу.

Хеш-таблиці часто застосовуються в базах даних, і, особливо, в мовних процесорах типу компіляторів і ассемблеров, де вони підвищують швидкість обробки таблиці ідентифікаторів. Як використання хешування в повсякденному житті можна навести приклади розподіл книг в бібліотеці по тематичним каталогам, упорядкування в словниках за першими літерами слів, шифрування спеціальностей у вищих навчальних закладах і т.д.

Основні поняття хешування даних

На малюнку схематично зображено хеш-таблицю. Вхідні дані проходять крізь хеш функцію у результаті отримуються ключі, які переміщується в комірки у таблиці.



Основні поняття хешування даних

Є певні правила, які мають виконуватись у хеш-функціях:

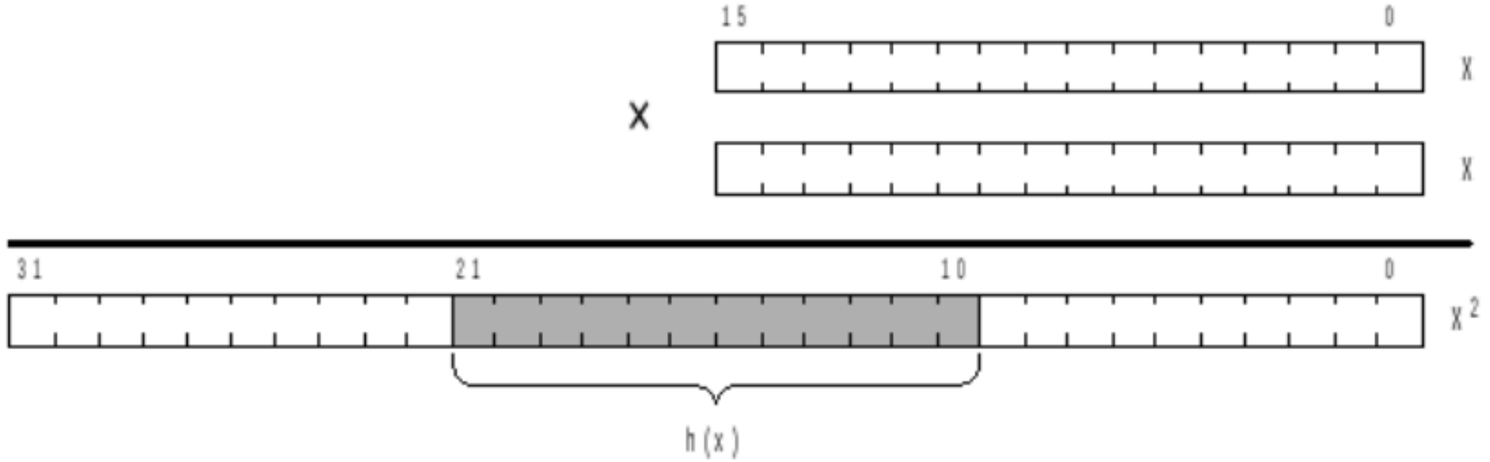
- завжди повертати одну й ту саму адресу для одного і того ж ключа
- використовувати всі адреси з однаковою ймовірністю, для того щоб не можна було сказати по віддаленості двох адрес одна від одного про розташування ключів
- не обов’язково повертати різні адреси для різних ключів це також випливає з того що через адресу не можна передати будь-яку інформацію про ключ
- швидко обчислювати адреси

Хеш функції на основі середини квадратів

Ключ підводиться до квадрату і береться кілька розрядів з середини ключа.

Тут і далі передбачається, що ключ спочатку округлюється до цілого числа, для здійснення з ним арифметичних операцій. Однак такий спосіб добре працює до моменту, коли немає великої кількості нулів зліва або справа.

Нехай є певне 16-ти розрядне число, тоді його квадрат займатиме 32 розряди.



Хеш функції на основі ділення

За Кнутом це один із двох варіантів, що добре зарекомендував себе по результатах багаточисельних тестів. Суть методу полягає в тому, що ми використовуємо залишок від ділення на M. Тобто при розмірі хеш таблиці M і значенні ключа K=5.

$$hash(K) = K \ mod \ M$$

Кунт звертає увагу, на той момент, що при M парне число значення hash(K) буде також буде парним і при непарному - непарним, це призводить до значного зміщення даних у багатьох файлах.

Даний метод підходить не для всіх значень.

- Якщо значення M ступінь 2, то $K \ mod \ M$ буде декілька цифр числа K, що розміщені з правої сторони і не будуть залежати від решти цифр.

- M не має бути кратно 3, бо при символьному ключі два з них будуть відрізнятися лише перестановкою символів і можуть давати числові значення із різницею кратною 3.

- Треба уникати M, що діляться на $r^k \pm a$, де k и a - невеликі числа, а $r - 64, \ 256$ чи 100 або інша основа системи числення, тому що залишок від ділення по модулю M буде проста суперпозиція цифр ключа. Тому треба вибирати M як просте число.

Хеш функції на основі множення

Для цього методу хешування використовується формула:

$$hash(K) = [M * ((C * K) \ mod \ 1)]$$

З формули видно, що ключ K множиться на деяку константу, що лежить у межах $0 < C < 1$. Після цього береться дробова частина цього виразу і множиться на M, і результат замінюється на найближче ціле менше число, щоб не вийти за границі хеш таблиці.

Якщо константа C обрана вірно, то можна домогтися дуже хороших результатів, однак, цей вибір складно зробити. Дональд Кунт зазначає, що множення може іноді виконуватися швидше ділення.

Хеш функції. Циклічний надлишковий код (CRC)

Основна ідея алгоритму CRC полягає у тому що K, тобто вхідну інформацію подають у вигляді величезного двійкового числа, ділять на константу поліном і використовують залишок від поділу в якості контрольної суми.

Існує каталог алгоритмів CRC, зі вказаними поліномами для кожного алгоритму, у таблиці наведено лише деякі із них. Число після CRC означає довжину отриманого хешу в бітах.

Алгоритм	Поліном
CRC-6/ITU	0x3
CRC-8	0x7
CRC-16	0x8005
CRC-16-CCITT	0x1021
CRC-32C	0x1EDC6F41

Хеш функції. Циклічний надлишковий код (CRC)

Найпростіша реалізація цього алгоритму це прямий поділ вхідного числа на поліном. Є базовий алгоритм обчислення контрольної суми CRC

1. Створюється масив, заповнений нулями, рівний по довжині розрядності полінома.

2. Оригінал тексту доповнюється нулями в молодших розрядах, в кількості, що дорівнює числу розрядів полінома.

3. В молодший розряд регістра заноситься один старший біт повідомлення, а з старшого розряду регістра висувається один біт.

4. Якщо висунутий біт дорівнює 1, то проводиться інверсія бітів (операція XOR, що виключає АБО) в тих розрядах регістру, які відповідають одиницям в поліномі.

5. Якщо в повідомленні ще є біти, система переходить до кроку 3.

Коли всі біти повідомлення надійшли в регістр і були оброблені цим алгоритмом, в регістрі залишається залишок від ділення, який і є контрольноюсумою CRC.

Хеш функції. Дайджест повідомлення (MD).

MD5 (Message Digest 5) — 128-бітний алгоритм хешування, розроблений професором Рональдом Л. Рівестом в 1991 році. Призначений для створення «відбитків» або «дайджестів» повідомлень довільної довжини. Прийшов на зміну MD4, що був недосконалим. Описаний в RFC 1321. З 2011 року відповідно RFC 6151 алгоритм вважається ненадійним.

Всі обчислення йдуть у 5 етапів. Та враховуючи, що з 2011 року RFC 6151 випустило документ, в якому алгоритм вважається ненадійним не будемо зупинятися на ньому детально.

Хеш функції. Універсальне хешування.

Під час універсального хешування вибір хеш-функції відбувається під час виконання програми випадковим чином з пулу хеш-функцій. Тобто при повторному виклику алгоритму з тими ж даними алгоритм може працювати вже зовсім по-іншому.

Колізії. Методи вирішення колізій

Враховуючи, що одна і та сама адреса може використовуватися для різних ключів можливі виникнення колізій.

Колізія — це ситуація, коли hash(K1) = hash(K2). У цьому випадку, необхідно знайти нове місце для зберігання даних.

Очевидно, що кількість колізій необхідно звести до мінімального значення. Хеш-функція повинна задовольняти двом вимогам:

- її обчислення повинно виконуватися дуже швидко;
- вона повинна мінімізувати число колізій.

Для усунення колізій існує декілька варіантів.

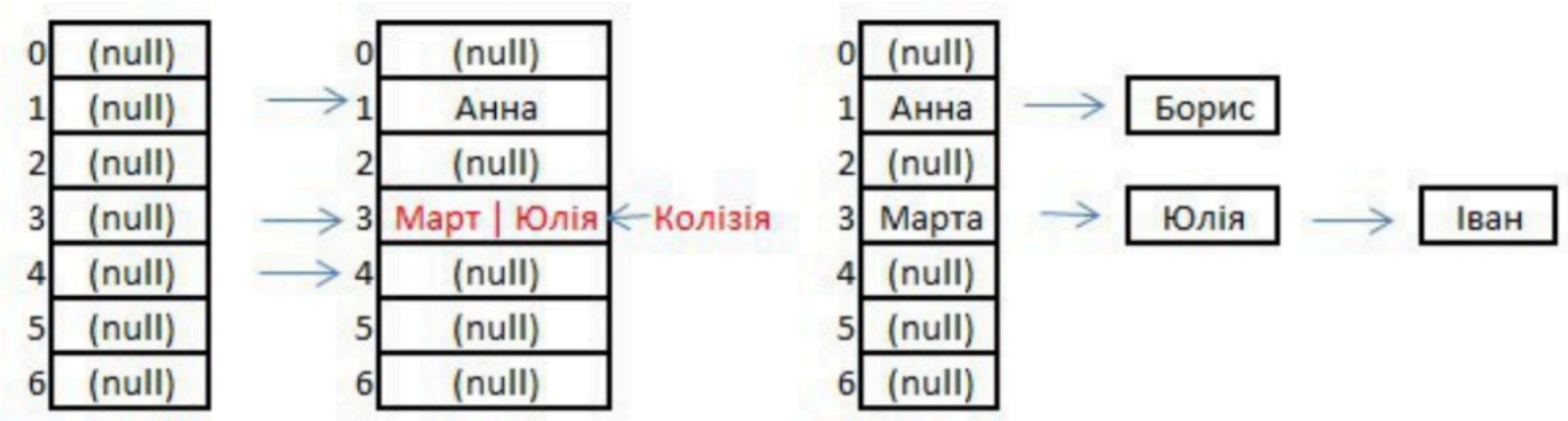
Колізії. Метод ланцюжків.

Хеш-таблиця складається із двох частин, а саме фактичної таблиці, де зберігаються дані, і хеш-функції, яка використовується для відображення ключів індексу до значень. У звичайних операціях хеш-таблиця оцінює вхідні дані, а потім присвоює цьому індексу ключ (розташування таблиці). Тепер, коли два різних ключа хешуються до того ж індексу в хеш-таблиці, вважається, що зіткнення відбулося.

Пам`ять обмежена, а системам потрібна неймовірна кількість пам`яті, щоб колізії не відбулися. Зіткнення зазвичай рівномірно розподіляються навколо хеш-таблиці. Це означає, що в теорії жодна позиція на хеш-таблиці не буде надмірно переповнена довгим списком зіткнень порівняно з іншими місцями в таблиці.

Колізії. Метод ланцюжків.

На малюнку показано випадки зіткнень в різних місцях таблиць для прикладу масив [Анна, Марта, Юлія, Борис, Іван].



Колізії. Метод ланцюжків.

До переваг можна віднести:

1. легкість реалізації.

2. теоретична нескінченність, за рахунок того. що хеш-таблиця ніколи не заповнюється, завжди можна додати більше елементів до ланцюга.

3. за рахунок 2 пункту використовується, коли невідомо, скільки і як часто дані можуть бути вставлені або видалені.

Із мінусів реалізації:

1. деякі частини хеш-таблиці ніколи не використовуються.

2. якщо ланцюг стає довгим, то час пошуку може стати O (n) у гіршому випадку.

3. через те, що ланцюжки зберігаються за допомогою зв’язаного списку падає ефективність кешування ланцюжка.

4. використовує додатковий простір для посилань.

Колізії. Метод відкритої адресації.

Підхід до хешування, відмінний від методу ланцюжків, був запропонований Петерсоном. Під час використання цього методу у будь-який момент розмір таблиці повинен бути більшим або рівним загальній кількості ключів. Бажано, використовувати лише 70% таблиці.

У процесі заповнення хеш-таблиці, при отриманні нового значення, його заносять у випадку, якщо його комірка ще не заповнена і значення None.

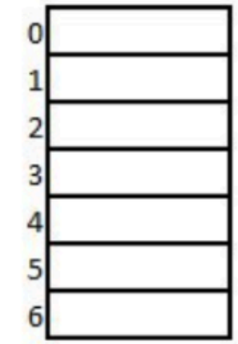
В іншому випадку значення заноситься в іншу комірку, яка обчислюється за допомогою певного алгоритму, але якщо і вона виявляється вже заповнена, то обчислюється ще одна ланка допоки не буде знайдена вільна комірка.

Колізії. Метод відкритої адресації.

Назва **відкрита адресація** відноситься до того, що адреса (розташування) елемента не визначається його хеш-значенням. Цей метод також називається **замкнутим хешуванням**; його не слід плутати з відкритим хешуванням або закритою адресацією.

Колізії. Метод відкритої адресації.

Проте важливим є кількість порівнять, якщо їх доводиться робити більше 3-4 то використання хеш-таблиці стає недоцільним і треба використати іншу хеш функцію для зменшення кількості порівнянь.



Ініціюємо пусту таблицю

Вставляємо "Анна"

Вставляємо "Іван" та "Юлія"

Вставляємо "Борис" Колізія виникає, вставляємо "Борис" у наступний вільний слот

Вставляємо "Саша", колізія виникає оскільки "Анна" на індексі 1, вставляємо у вільний наступний

Вставляємо "Галина" та "Микола"

Колізії. Метод відкритої адресації.

За рахунок того, що існує декілька варіантів обрахування вільного місця алгоритм відритої адресації може бути винонаний декількома методами.

Метод лінійних проб. Це найпростіший метод відкритої адресації під чає якого новий ключ записується у наступну порожню комірку. Тобто якщо комірка hash(K) вже є зайтяною переходимо до наступної
(hash(K) + 1). Якщо і друга є зайнятою переходимо до третьої (hash(K) + 2). При досягненні кінця геш таблиці пошук починається спочатку.

Недоліком даного методу є утворення групувань заповнених комірців. Якщо значне число послідовних елементів таблиці зайняте, то ймовірність попадання нового елемента в першу вільну комірку після цієї групи зростає пропорційно кількості елементів в групі. Такі групи осередків збільшуються і в результаті порушується рівномірність розподілення ключів, що призводить до втрати ефективності пошуку і включення нових елементів.

Колізії. Метод відкритої адресації.

Метод квадратичних проб. Цей метод може бути більш ефективним ніж метод лінійних проб, оскільки він краще уникає проблеми кластеризації. В даному випадку функція гешування виглядає

$$hash(K)_i = hash(K)_0 + i^2$$

До мінусів можна віднести можливість існування вільних комірок в таблиці, але вони не можуть бути знайдені при повторному хешуванні. Зазвичай максимальна допустима кількість спроб розміщення елементу при використанні цього методу визначається константою або залежить від розміру таблиці. Втім, якщо розмір таблиці - просте число, то гарантується заповнення останньої хоча б наполовину.

Колізії. Метод відкритої адресації.

Метод подвійного хешування.

У ідеальному випадку операції вставки, видалення і пошуку виконуються за O (1), в гіршому - за O(m), що не відрізняється від звичайного лінійного вирішення колізій.

Однак в середньому, при грамотному виборі хеш-функцій, подвійне хешування видаватиме кращі результати, за рахунок того, що ймовірність збігу значень відразу двох незалежних хеш-функцій нижче, ніж однієї.

Для видалення елементу з таблиці створюється новий масив deleted типів bool, рівний за величиною масиву table. При видаленні елемент позначається як видалений, а при додаванні в цю комірку заміщається новим. При пошуку, крім рівності ключів перевіряється і якщо елемент помічений як видалений, пошук йде далі.

Колізії. Метод відкритої адресації.

Метод подвійного хешування.

При подвійному хешуванні використовуються дві незалежні хеш-функції перша - *hash1(K)* і друга - *hash2(K)*.

Першою перевіряється комірка за адресою першої хеш функції hash1(K), якщо вона вже зайнята, то розглядається (hash1(K)+ hash2(K))modM, потім (hash1(K) + 2·hash2(K))modM і так далі. У загальному випадку йде перевірка послідовності осередків (hash1(K) + i·hash2(K))modM де i = (0,1, ..., m-1), M - розмір таблиці, n mod m - залишок від ділення n на m.

Колізії. Порівняльна характеристика.

Таблиця 2. Порівняння методі ланцюжків та відкритої адресації.

	Метод ланцюжків	Відкрита адресація
Метод вирішення колізій	Використання зовнішньої структури даних	Використання самої геш-таблиці
Витрати пам’яті	Розмір покажчика накладних витрат на запис (збереження головок списку в таблиці)	Немає
Залежність продуктивності від коефіцієнта навантаження таблиці	Прямо пропорційно	Пропорційно загрузці / (1 - коефіцієнт навантаження)
Можливість зберігати більше елементів, ніж розмір хеш-таблиці	Так	Ні. Крім того, рекомендується зберігати коефіцієнт завантаження таблиці нижче 0.7
Вимоги до функцій хеш-пам'яті	Рівномірний розподіл	Рівномірний розподіл з уникненням кластеризації
Можливість ручного видалення	Добре видалення	Забороняється у хеш-таблиці і можлива за допомогою запису "DELETED"
Реалізація	Проста	Коректна реалізація хеш-таблиці на основі відкритої адресації досить складна

Дякую за увагу!