ЛЕКЦІЯ 9. ХЕШ-ТАБЛИЦІ

9.1. Таблиці з прямою адресацією

Хеш-таблиця представляє собою узагальнення звичайного масиву. Можливість прямої індексації елементів звичайного масиву дозволяє доступ до довільної позиції в масиві за час O(1).

Загалом *пряма індексація* представляє собою технологію, яка добре працює для невеликих множин ключів. Припустимо, що застосуванню потрібна динамічна множина, кожний елемент якої має клюз з множини $U = \{0,1,\ldots m-1\}$, де m не дуже велике. Крім того, припускається, що жодні два елементи не мають однакових ключів.

Для представлення динамічної множини використовується масив, або таблиця з прямою адресацією, який позначається як $T[0\ldots m-1]$, кожна позиція, або комірка, якого відповідає ключу з простору ключів U. На рис. 9.1 представлений цей підхід. Комірка k вказує на елемент множини з ключем k. Якщо множина не містить елементу з таким ключем, то $T[k] = \text{NULL}$. На рисунку кожний ключ простору $U = \{0, 1,\ldots,9\}$ відповідає індексу таблиці. Множина реальних ключів $K = \{2, 3, 5, 8\}$ визначає комірки таблиці, які містять покажчики на елементи. Решта комірок містять значення NULL.

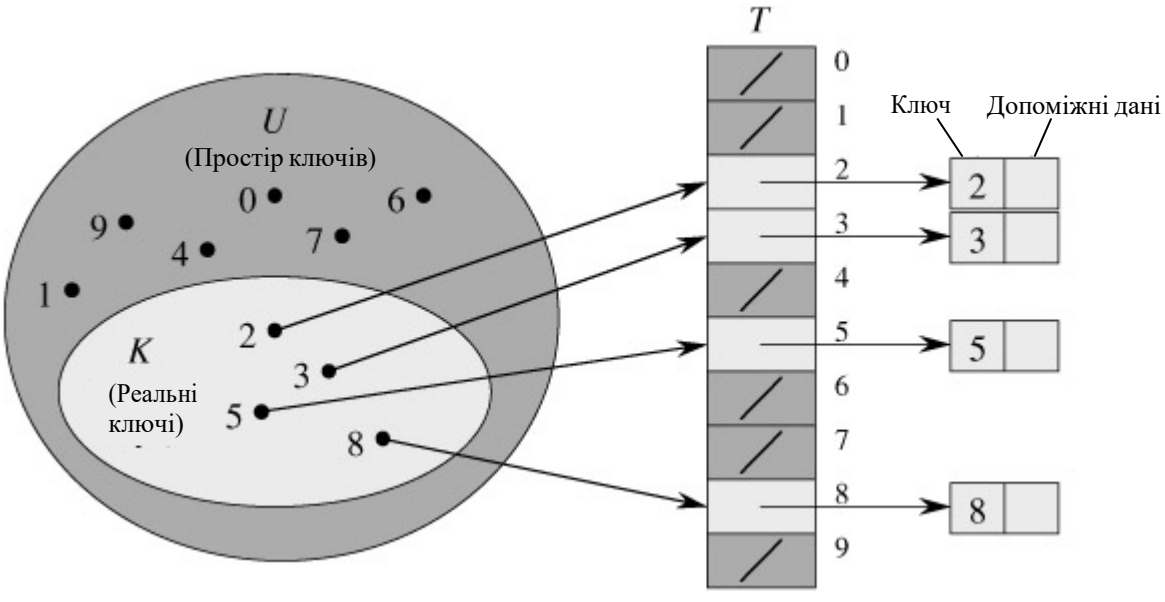

Рис. 9.1. Динамічна множина із використанням таблиці з прямою адресацією.

Реалізація основних операцій в масиві тривіальна.

```
DirectAddressSearch(T, k)
1    return T[k]

DirectAddressInsert(T, x)
1    T[key[x]] = x

DirectAddressDelete(T, x)
1    T[key[x]] = NULL
```

Лістинг 9.1. Процедури, які реалізують операції роботи з масивами.

Кожна з наведених операцій дуже швидка: час їх роботи дорівнює O(1).

В деяких застосунках елементи динамічної множини можуть зберігатись безпосередньо в таблиці з прямою адресацією. Тобто замість зберігання ключів та допоміжних даних елементів в об'єктах, які є зовнішніми по відношенню до таблиці з прямою адресацією, а в таблиці – вказівників на ці об'єкти, ці об'єкти можна зберігати безпосередньо в комірках таблиці (що призводить до економії пам'яті).

9.2. Хеш-таблиці

Недолік прямої адресації очевидний: якщо простір ключів U великий, то зберігання таблиці T розміром $|U|$ непрактично, а для деяких задач навіть неможливо. Крім того, множина K реальних ключів може бути малою по відношенню до U, а в цьому випадку пам'ять, яка виділена під T, здебільшого витрачається дарма.

Коли множина K ключів, які зберігаються у словнику, значно менша простору U, хеш-таблиця вимагає значно менше місця, чим таблиця з прямою адресацією. Точніше кажучи, вимоги до пам'яті можуть бути знижені до $\Theta(|K|)$, при цьому час пошуку елементу в хеш-таблиці лишається рівним O(1). Потрібно тільки відзначити, що це межа середнього часу пошуку, в той час як у випадку таблиці з прямою адресацією ця межа справедлива для найгіршого випадку.

У випадку прямої адресації елемент з ключем k зберігається у комірці k. При хешуванні цей елемент зберігається в комірці $h(k)$, тобто тут використовується *хеш-функція* h для обчислення комірки для даного ключа k.

Функція h відображає простір ключів U на комірки *хеш-таблиці* $T[0\ldots m-1]$:

$$h: U \to T\{0,1,\ldots m-1\}.$$

Ми говоримо, що елементи з ключем k хешується в комірку $h(k)$; величина $h(k)$ називається хеш-значенням ключа k. На рис. 9.2 представлена основна ідея хешування. Мета хеш-функції полягає в тому, щоб зменшити робочий діапазон індексів масиву, а замість $|U|$ значень ми можемо обійтись лише m значеннями. Відповідно знижуються вимоги до об'єму пам'яті.

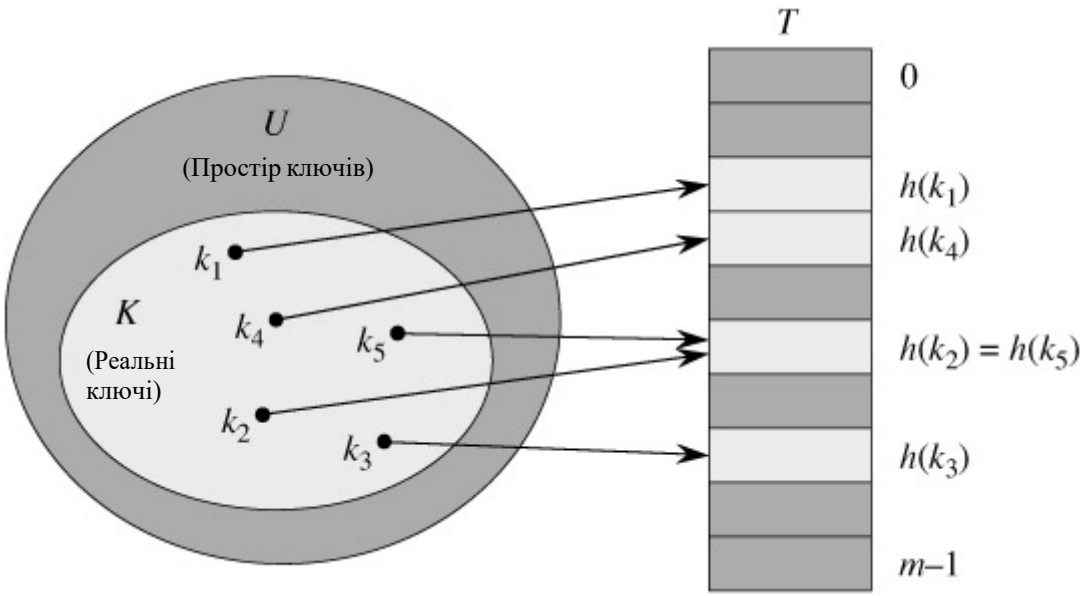

Рис. 9.2. Використання хеш-функції для відображення ключів у комірки хеш-таблиці.

Проте тут є одна проблема: два ключа можуть мати одне й те саме хеш-значення. Так ситуація називається *колізією*. На щастя є ефективні технологія для розв'язання конфліктів, які виникають під час колізій.

Звісно, ідеальним рішенням було би повне уникнення колізій. Цього можна досягнути шляхом підбора відповідної хеш-функції. Одна з ідей полягає в тому, щоб зробити функцію h випадковою. Зрозуміло, що хеш-функція мусить бути детермінованою і для одного й того самого значення k завжди давати одне й те саме значення $h(k)$. Однак, оскільки $|U| > m$, повинні існувати як мінімум два ключа, які мають однакове хеш-значення. Таким чином, повністю уникнути колізій

принципово неможливо, і хороша хеш-функція може тільки мінімізувати кількість колізій.

9.3. Уникнення колізій за допомогою ланцюгів

За допомогою методу ланцюгів всі елементи, які хешуються в одну й ту саму комірку, об'єднуються у зв'язаний список, як це показано на рис. 9.3. Комірка *j* містить вказівник на заголовок списку всіх елементів, хеш-значення ключа яких дорівнює *j*; якщо таких елементів немає, комірка містить значення NULL. На рис. 9.3 наведене розв'язання колізій, які виникають через те, що $h(k_1) = h(k_4)$, $h(k_5) = h(k_2) = h(k_7)$ та $h(k_8) = h(k_6)$.

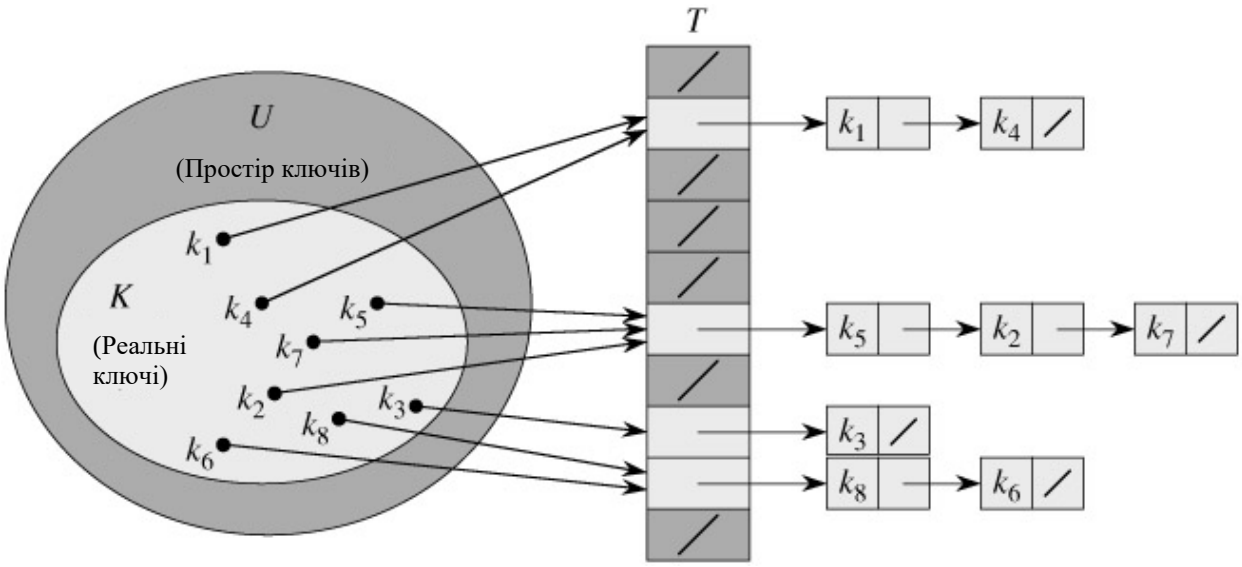

Рис. 9.3. Розв'язання колізій за допомогою ланцюгів.

Словникові операції в хеш-таблиці із використанням ланцюгів для розв'язання колізій реалізуються дуже просто:

```
ChainedHashInsert(T, x)
1    Вставити x в заголовок списку T[h(key[x])]

ChainedHashSearch(T, k)
1    Пошук елементу з ключем k в списку T[h(k)]

ChainedHashDelete(T, x)
1    Видалення x зі списку T[h(key[x])]
```

Лістинг 9.2. Процедури, які реалізують операції роботи з хеш-таблицями.

Час, необхідний для вставки в найгіршому випадку, дорівнює O(1).

Процедура вставки виконується дуже швидко, адже передбачається, що елемент для вставки відсутній у таблиці. При необхідності це припущення може бути перевірено виконанням пошуку перед вставкою. Час роботи пошуку в найгіршому випадку пропорційний довжині списку. Видалення елементу може бути виконане за час O(1) при використанні двозв'язаних списків.

Наскільки висока ефективність хешування з ланцюгами? Зокрема, скільки часу потрібно для пошуку елементу з даним ключем?

Нехай ми маємо хеш-таблицю T з m комірками, в яких зберігаються n елементів. Визначимо *коефіцієнт заповнення* таблиці T як $\alpha = n/m$, тобто як середню кількість елементів, які зберігаються в одному ланцюгу. Подальший аналіз буде ґрунтуватись на значенні величини α, яка може бути менше, дорівнювати або бути більше одиниці.

В найгіршому випадку хешування з ланцюгами поводить себе досить неприємно: всі n ключів хешуються в одну й ту саму комірку, створюючи при цьому список довжиною n. Таким чином, час пошуку в найгіршому випадку дорівнює $\Theta(n)$ плюс час обрахунку хеш-функції, що нічим не краще використання зв'язаного списку для збереження всіх n елементів. Зрозуміло, що використання хеш-таблиць в найгіршому випадку немає сенсу.

Середня продуктивність хешування залежить від того, наскільки добре хеш-функція h розподіляє множину ключів для зберігання по m коміркам в середньому. Ми розглянемо це питання детальніше згодом, а зараз припустимо, що всі елементи хешуються по комірках рівномірно та незалежно, і назвемо це припущення „*простим рівномірним хешуванням*".

Позначимо довжини списків $T[j]$ для $j = 0, 1, \ldots, m-1$ як n_j, так що

$$n = n_0 + n_1 + \ldots + n_{m-1},$$

а середнє значення n_j дорівнює $E\left[n_j\right] = \alpha = n/m$.

Будемо вважати, що хеш-значення $h(k)$ може бути обраховане за час O(1), так що час, необхідний для пошуку елементу з ключем k, лінійно залежить від довжини $n_{h(k)}$ списку $T[h(k)]$. Розглянемо математичне сподівання кількості елементів, яке повинно бути перевірене алгоритмом пошуку (тобто кількість

елементів в списку $T[h(k)]$, які перевіряються на рівність їх ключів величині k). Ми повинні розглянути два випадки: по-перше, коли пошук невдалий і в таблиці немає елементів з ключем k, і, по-друге, коли пошук закінчується вдало і в таблиці виявлений елемент з ключем k.

Теорема 9.1. В хеш-таблиці з розв'язком колізій методом ланцюгів математичне сподівання часу невдалого пошуку у припущенні простого рівномірного хешування дорівнює $\Theta(1+\alpha)$.

Доведення. У припущенні простого рівномірного хешування довільний ключ k, який ще не знаходиться у таблиці, може бути розміщений з однаковою ймовірністю в будь-яку з m комірок. Математичне сподівання часу невдалого пошуку ключа k дорівнює часу пошуку до кінця списку $T[h(k)]$, математичне сподівання довжини якого $E\left[n_j\right]=\alpha$. Таким чином, при невдалому пошуку математичне сподівання кількості елементів, які перевіряються, дорівнює α, а загальний час, необхідний для пошуку, включаючи час обрахунку хеш-функції $h(k)$, дорівнює $\Theta(1+\alpha)$. ■

Вдалий пошук дещо відрізняється від невдалого, оскільки ймовірність пошуку в списку відмінна для різних списків та пропорційна кількості елементів, які містяться в ньому. Тим не менш, і в цьому випадку математичне сподівання часу пошуку лишається рівним $\Theta(1+\alpha)$.

Теорема 9.2. В хеш-таблиці з розв'язком колізій методом ланцюгів математичне сподівання часу вдалого пошуку у припущенні простого рівномірного хешування дорівнює $\Theta(1+\alpha)$.

Доведення. Будемо вважати, що шуканий елемент з однаковою ймовірністю може бути довільним елементом, який зберігається в таблиці. Кількість елементів, які перевіряються в процесі вдалого пошуку елементу x, на 1 більше, ніж кількість елементів, які знаходяться у списку перед x. Елементи, які знаходяться в списку до x, були вставлені в список після того, як елемент x був збережений в таблиці, адже нові елементи додаються у початок списку. Для того щоб знайти математичне сподівання кількості елементів, які перевіряються, візьмемо середнє

значення по всім елементам таблиці, яке дорівнює 1 плюс математичне сподівання кількості елементів, доданих у список після шуканого. Нехай x_i означає i-й елемент, який вставлений у таблицю ($i = 1, 2,\ldots,n$), а $k_i = key[x_i]$. Визначимо для ключів k_i та k_j індикаторну випадкову величину $X_{ij} = I\left\{h\left(k_i\right) = h\left(k_j\right)\right\}$. За припущенням простого рівномірного хешування, $P\left(h\left(k_i\right) = h\left(k_j\right)\right) = 1/m$ і, відповідно, $E\left[X_{ij}\right] = 1/m$. Таким чином, математичне сподівання кількості елементів, які перевіряються, у випадку вдалого пошуку дорівнює:

$$E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}X_{ij}\right)\right] = \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}E\left[X_{ij}\right]\right) = \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right) = 1+\frac{1}{nm}\sum_{i=1}^{n}(n-i) =$$

$$= 1+\frac{1}{nm}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right) = 1+\frac{1}{nm}\left(n^2-\frac{n(n+1)}{2}\right) = 1+\frac{n-1}{m} = 1+\frac{\alpha}{2}-\frac{\alpha}{2n}$$

Відповідно, повний час, необхідний для проведення вдалого пошуку, включаючи час на обрахунок хеш-функції, складає $\Theta\left(2+\alpha/2-\alpha/2n\right) = \Theta\left(1+\alpha\right)$. ■

Що означає проведений аналіз? Якщо кількість комірок в хеш-таблиці, як мінімум, пропорційна кількості елементів, які зберігаються в ній, то $n = \mathrm{O}(m)$ і, відповідно, $\alpha = n/m = \mathrm{O}\left(m\right)/m = \mathrm{O}\left(1\right)$, і значить пошук елементу в хеш-таблиці в середньому потребує сталий час. Оскільки в найгіршому випадку вставка елементу в хеш-таблиці займає O(1) часу (як і видалення елементу при використанні двонаправлених списків), можна зробити висновок, що всі словникові операції в хеш-таблиці в середньому виконується за час O(1).

9.4. Хеш-функції

Якісна хеш-функція задовольняє (наближено) припущення простого рівномірного хешування: для кожного ключа рівно ймовірне розміщення в будь-яку з m комірок, незалежно від хешування інших ключів. Нажаль, цю умова зазвичай неможливо перевірити, оскільки, як правило, розподілення ймовірностей, згідно з яким поступають нові ключі, невідоме; крім того, вставлені ключі можуть й не бути незалежними.

Іноді розподілення ймовірностей виявляється відомим. Наприклад, якщо відомо, що ключі представляють собою випадкові дійсні числа, які рівномірно розподілені в діапазоні $0 \le k < 1$, то хеш-функція $h(k) = \lfloor km \rfloor$ задовольняє умові простого рівномірного хешування.

На практиці при побудові якісних хеш-функцій часто використовується різні евристичні методики. В процесі побудови значно допомагає інформація про розподіл ключів. Розглянемо, наприклад, таблицю символів компілятора, в який ключами слугують символьні рядки, які представляють ідентифікатори в програмі. Часто в одній програмі зустрічаються схожі ідентифікатори, наприклад, `pt` та `pts`. Якісна хеш-функція повинна мінімізувати шанси потрапляння цих ідентифікаторів в одну комірку хеш-таблиці.

При побудові хеш-функції гарним підходом є підбір функції таким чином, щоб вона ніяк не корелювала із закономірностями, яким можуть підпорядковуватись існуючі дані. Також деякі застосування хеш-функцій можуть накладати додаткові вимоги, окрім вимог простого рівномірного хешування. Наприклад, можна вимагати, щоб „близькі" в деякому сенсі ключі давали далекі хеш-значення.

Для більшості хеш-функцій простір ключів представляється множиною невід'ємнх цілих чисел $N = \{0,1,2,\ldots\}$. Якщо ж ключі не є цілими невід'ємними числами, то можна знайти спосіб їх інтерпретації як таких. Наприклад, рядок символів може розглядатись як ціле число, яке записане у відповідній системі числення. Так, ідентифікатор `pt` можна розглядати як пару десяткових чисел (112, 116), оскільки в ASCII-наборі символів `p` = 112 та `t` = 116. Розглядаючи `pt` як число в системі числення з основою 128, ми бачимо, що воно відповідає значенню $112 \cdot 128 + 116 = 14452$. В конкретних застосунках зазвичай не виявляється надто складно розробити метод для представлення ключів у вигляді цілих (можливо, великих) чисел. Далі ми будемо вважати, що всі ключі представлені як цілі невід'ємні числа.

Розглянемо наступні два методи побудови хеш-функцій: метод ділення та метод множення.

Побудова хеш-функції *методом ділення* полягає у відображені ключа k в одну з комірок шляхом отримання остачі від ділення k на m, тобто хеш-функція має вигляд $h(k) = k \bmod m$. Наприклад, якщо хеш-таблиця має розмір $m = 12$, а значення ключа $k = 100$, то $h(k) = 4$. Оскільки для обчислення хеш-функції потрібна тільки одна операція ділення, хешування методом ділення вважається достатньо швидким.

При використанні даного методу зазивай намагаються уникнути деяких значень m. Наприклад, m не повинно бути степенем 2, оскільки якщо $m = 2^p$, то $h(k)$ представляє собою просто p молодших бітів числа k. Якщо тільки завчасно не відомо, що всі набори молодших p бітів ключів рівноймовірні, краще будувати хеш-функції таким чином, щоб її результат залежав від усіх бітів ключа.

Часто добрі результати можна отримати, якщо обирати в якості значення m просте число, достатньо далеке від степеня двійки. Припустимо, наприклад, що ми хочемо створити хеш-таблицю із розв'язком колізій методом ланцюгів для збереження $n = 2000$ символьних рядків, розмір символів в яких дорівнює 8 бітам. Нас влаштовує перевірка у середньому трьох елементів при невдалому пошуку, так що ми обираємо розмір таблиці рівним $m = 701$. Число 701 обрано як просте, яке близьке до 2000/3.

Побудова хеш-функції *методом множення* виконується в два етапи. Спочатку ключ k помножується на сталу $0 < A < 1$ і береться дробова частина отриманого добутку. Потім це значення помножується на m і результат замінюється на найближче менше ціле число, тобто:

$$h(k) = \lfloor m(kA \bmod 1) \rfloor,$$

де вираз „$kA \bmod 1$" означає отримання дробової частини добутку kA.

Перевага методу множення полягає в тому, що значення m перестає бути критичним. Зазвичай величина m із міркувань зручності реалізації функції обирається рівною степеню 2. Нехай ми маємо комп'ютер із розміром слова w бітів та k розміщується в одне слово. Обмежимо можливі значення константи A виглядом $s / 2^w$, де s – ціле число з діапазону $0 < s < 2^w$. Тоді ми спочатку помножуємо k на w-бітове ціле число $s = A \cdot 2^w$. Результат представляє собою $2w$-бітове число $r_1 2^w + r_0$

, де r_1 – старше слово добутку, а r_0 – молодше. Старші p бітів числа r_0 представляють собою шукане p-бітове хеш-значення (рис. 9.4).

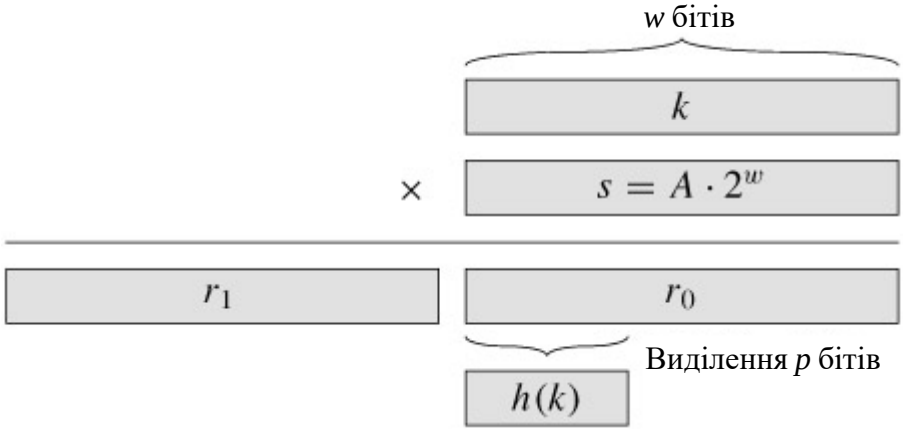

Рис. 9.4. Хешування методом множення.

Хоч описаний метод працює з довільними значенням константи A, деякі значення дають кращі результати у порівнянні з іншими. Оптимальний вибір залежить від характеристик даних, що хешуються. Так, непогані результати дає значення

$$A \approx \left(\sqrt{5}-1\right)/2 \approx 0,6180339887\ldots$$

Наприклад, при $k = 123456$, $p = 14$, $m = 2^{14} = 16384$ та $w = 32$, оберемо значення A у вигляді 2^w, яке найближче до $\left(\sqrt{5}-1\right)/2$, так що $A = 2654435769/2^{32}$. Тоді

$$k \cdot s = 327706022297664 = \left(76300 \cdot 2^{32}\right) + 17612864,$$

і, відповідно, $r_1 = 76300$ та $r_0 = 17612864$. Старші 14 бітів числа r_0 дають хеш-значення $h(k) = 67$.

У наведених способах побудови хеш-функцій є один суттєвий недолік. Якщо зловмисник буде свідомо обирати ключі для хешування за допомогою конкретної хеш-функції, то він зможе підібрати n значень, які будуть хешуватись в одну й ту саму комірку таблиці, що призведе до середнього часу вибірки $\Theta(n)$. Таким чином, будь-яка фіксована хеш-функція стає вразливою, і єдиний ефективний вихід з цієї ситуації – випадковий вибір хеш-функції, який не залежить від того, з якими саме ключами їй доведеться працювати. Такий підхід, який називається *універсальним*

хешуванням, гарантує добру якість в середньому, не залежно від того, які дані будуть обрані зловмисниками.

Головна ідея універсального хешування полягає у випадковому виборі хеш-функції з деякого ретельно відібраного класу функцій на початку роботи програми. Як і у випадку швидкого сортування, рандомізація гарантує, що одні й ті самі вхідні дані не можуть постійно давати найгіршу поведінку алгоритму. В силу рандомізації алгоритм буде працювати кожен раз по різному, навіть для одних й тих самих вхідних даних, що гарантує високу середню продуктивність. Для більше детальної інформації по універсальному хешуванню рекомендуємо звернутись до літератури, наведеної у кінці даного матеріалу.

9.5. Відкрита адресація

При використанні *методу відкритої адресації* всі елементи зберігаються безпосередньо в хеш-таблиці, тобто кожний запис таблиці містить або елемент динамічної множини, або значення NULL. При пошуку елемента ми систематично перевіряємо комірки таблиці до тих пір, доки не знайдемо шуканий елемент або поки не переконаємось у його відсутності в таблиці. Тут, на противагу методу ланцюгів, немає ані списків, ані елементів, які зберігаються поза таблицею. Таким чином, в методі відкритої адресації хеш-таблиця може виявитись заповненою, що унеможливить додавання нових елементів; коефіцієнт заповнення α не може бути більшим за 1.

Для виконання вставки при відкритій адресації ми послідовно перевіряємо комірки хеш-таблиці до тих пір, доки не знайдемо порожню комірку, в яку розміщується новий ключ. Замість фіксованого порядку дослідження комірок $0, 1,\ldots,m-1$ (для чого потрібний час $\Theta(n)$), послідовність досліджуваних комірок залежить від ключа, який вставляється у таблицю. Для визначення досліджуваних комірок хеш-функція розширюється, включаючи в якості другого аргументу номер дослідження (який починається з 0). В результаті хеш-функція стає наступною:

$$h: U \times \{0,1,\ldots,m-1\} \to \{0,1,\ldots,m-1\}.$$

У методі відкритої адресації потрібно, що для кожного ключа k послідовність

досліджень

$$\left[h(k,0),h(k,1),\ldots,h(k,m-1)\right]$$

представляла собою перестановку множини $\{0,1,\ldots m-1\}$, щоб в кінцевому випадку можна було переглянути всі комірки хеш-таблиці. У наведеному нижче псевдокоді припускається, що елементи в таблиці *T* представляють собою ключі без додаткової інформації; ключ *k* тотожній елементу, який містить ключ *k*. Кожна комірка містить або ключ, або значення NULL (якщо вона не заповнена):

```
HashInsert(T, k)
1    i = 0
2    repeat j = h(k, i)
3         if T[j] = NULL
4              then T[j] = k
5                   return j
6              else i = i + 1
7    until i = m
8    error "Хеш-таблиця переповнена"
```

Лістинг 9.3. Процедура додавання елементу до відкритої адресації.

Алгоритм пошуку ключа *k* досліджує ту саму послідовність комірок, що алгоритм вставки. Таким чином, якщо при пошуку зустрічається порожня комірка, пошук завершується невдало, оскільки ключ *k* повинен був бути вставлений в цю комірку у послідовності досліджень, але ніяк не після неї. Процедура `HashSearch()` отримує в якості вхідних параметрів хеш-таблицю *T* та ключ *k* і повертає номер комірки, яка містить ключ *k* (або значення NULL, якщо ключ в хеш-таблиці не знайдений):

```
HashSearch(T, k)
1    i = 0
2    repeat j = h(k, i)
3         if T[j] = k
4              then return j
5         i = i + 1
6    until T[j] = NULL або i = m
7    return NULL
```

Лістинг 9.4. Процедура пошуку ключа у відкритій адресації.

Процедура видалення з хеш-таблиці із відкритою адресацією достатньо складна. При видаленні ключа з комірки *i* ми не можемо просто помітити її значенням NULL. Зробивши так, ми можемо зробити неможливим вибірку ключа *k*, в процесі вставки якого досліджувалась і виявилась зайнятою комірка *i*. Одне з рішень полягає в тому, щоб помічати такі комірки спеціальним значенням DELETED замість NULL. При цьому ми повинні трохи змінити процедуру `HashInsert()`, з тим щоб вона розглядала таку комірку як порожню та могла вставляти в неї новий ключ. В процедурі `HashSearch()` жодних змін робити не потрібно, оскільки ми просто пропускаємо такі комірки при пошуку та досліджуємо наступні комірки під час пошуку в послідовності. Однак при використанні спеціального значення DELETED час пошуку перестає залежати від коефіцієнту заповнення α, і тому, як правило, при необхідності видалення з хеш-таблиці в якості методу розв'язання колізій обирається метод ланцюгів.

У подальшому ми будемо виходити із припущення *рівномірного хешування*, тобто ми припускаємо, що для кожного ключа в якості послідовності досліджень рівноймовірні всі $m!$ перестановок множини $\{0,1,\ldots m-1\}$. Рівномірне хешування представляє собою узагальнення визначеного раніше простого рівномірного хешування, яка полягає в тому, що тепер хеш-функція дає не одне значення, а цілу послідовність досліджень. Реалізація істинно рівномірного хешування достатньо важка, але на практиці використовуються прийнятні апроксимації.

Для обчислення послідовності досліджень для відкритої адресації зазвичай використовуються три методи: *лінійне дослідження*, *квадратичне дослідження* та *подвійне хешування*. Ці методи гарантують, що для кожного ключа k $\left[h(k,0),h(k,1),\ldots,h(k,m-1)\right]$ є перестановкою $\{0,1,\ldots m-1\}$. Однак ці методи не задовольняють припущенню про рівномірне хешування, адже жоден з них не в змозі згенерувати більше m^2 різних послідовностей (замість $m!$, які потрібні для рівномірного хешування). Найбільша кількість послідовностей досліджень дає подвійне хешування і, як слід, найкращі результати.

Лінійне дослідження. Нехай задана звичайна хеш-функція

$h' : U \to T\{0,1,\ldots m-1\}$, яку будемо надалі йменувати *допоміжною хеш-функцією*. Метод лінійного дослідження для обчислення послідовності досліджень використовує хеш-функцію

$$h(k,i) = \left(h'(k) + i\right) \bmod m,$$

де i приймає значення від 0 до $m-1$. Для заданого ключа k першою досліджуваною коміркою є $T\left[h'(k)\right]$, тобто комірка, яку дає допоміжна хеш-функція. Далі досліджуються комірки $T\left[h'(k)+1\right]$, $T\left[h'(k)+2\right],\ldots,T\left[m-1\right]$, а потім переходять до $T[0]$, $T[1]$, і далі до $T\left[h'(k)-1\right]$. Оскільки початкова досліджувана комірка однозначно визначає всю послідовність досліджень в цілому, всього є m різних послідовностей.

Лінійне дослідження легко реалізується, однак з ним пов'язана проблема первинної кластеризації, яка полягає у створенні довгих послідовностей зайнятих комірок, що, саме по собі, збільшує середній час пошуку. Кластери виникають через те, що ймовірність заповнення порожньої комірки, якій передують i заповнених комірок, дорівнює $(i+1)/m$. Таким чином, довгі серії заповнених комірок мають тенденцію до все більшого подовження, що призводить до збільшення середнього часу пошуку.

Квадратичне дослідження використовує хеш-функцію вигляду

$$h(k,i) = \left(h'(k) + c_1 i + c_2 i^2\right) \bmod m,$$

де h' – допоміжна хеш-функція, c_1 та $c_2 \neq 0$ – допоміжні константи, а i приймає значення від 0 до $m-1$. Початкова досліджувана комірка – $T\left[h'(k)\right]$; решта досліджуваних позицій зміщені відносно неї на величину, які описуються квадратичною залежністю від номеру дослідження i. Цей метод працює значно краще лінійного дослідження, але для того, щоб дослідження охопило всі комірки, необхідний вибір спеціальних значень c_1, c_2 та m. Окрім того, якщо два ключі мають одну й ту саму початкову позицію дослідження, то однакові й послідовності досліджень у цілому, адже з $h_1(k,0) = h_2(k,0)$ слідує $h_1(k,i) = h_2(k,i)$. Ця

властивість приводить до більш м'якої вторинної кластеризації. Як і у випадку лінійного дослідження, початкова комірка визначає всю послідовність, тому всього використовуються m різних послідовностей досліджень.

Подвійне хешування представляє собою один з найкращих методів використання відкритої адресації, оскільки отримані при цьому перестановки мають багато характеристик випадкового згенерованих перестановок. Подвійне хешування використовує хеш-функцію наступного вигляду:

$$h(k,i)=\left(h_1(k)+ih_2(k)\right)\bmod m,$$

де h_1 та h_2 – допоміжні хеш-функції. Початкове дослідження виконується в позиції $T\left[h_1(k)\right]$, а зміщення кожної з наступних досліджуваних комірок відносно попередньої дорівнює $h_2(k)$ по модулю m. Відповідно, на відміну від лінійного та квадратичного дослідження, в даному випадку послідовність досліджень залежить від ключа k по двох параметрах – в плані вибору початкової досліджуваної комірки та відстані між сусідніми досліджуваними комірками.

На рис. 9.5 показаний приклад вставки при подвійному хешуванні. Тут наведена хеш-таблиця розміром 13 комірок, в якій використовуються допоміжні хеш-функції $h_1(k)=k\bmod 13$ та $h_2(k)=1+(k\bmod 11)$. Так як $14\equiv 1(\bmod 13)$ та $14\equiv 3(\bmod 11)$, ключ 14 вставляється в порожню комірку 9, після того як при дослідженні комірок 1 та 5 з'ясовується, що ці комірки зайняті.

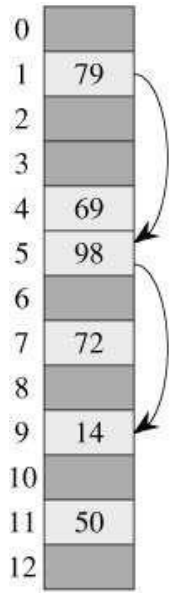

Рис. 9.5. Вставка елементу при подвійному хешуванні.

Для того щоб послідовність досліджень могла охопити всю таблицю, значення $h_2(k)$ повинно бути взаємно простим із розміром хеш-таблиці m. Зручний спосіб забезпечити виконання цієї умови полягає у виборі числа m, рівного ступеню 2, і розробці хеш-функції h_2 таким чином, щоб вона повертала тільки непарні значення. Ще один спосіб полягає у використанні в якості m простого числа і побудові хеш-функції h_2 такою, щоб вона завжди повертала натуральні числа, менші m. Наприклад, можна обрати просте число в якості m, а хеш-функції такими:

$$h_1(k) = k \bmod m,$$
$$h_2(k) = 1 + (k \bmod m'),$$

де m' повинно бути трохи менше m (наприклад, $m - 1$). Скажімо, якщо $k = 123456$, $m = 701$, а $m' = 700$, то $h_1(k) = 80$ та $h_2(k) = 257$, так що першою досліджуваною коміркою буде 80-а комірка, а потім буде досліджуватись кожна 257-а (за модулем m) комірка, доки не буде знайдена порожня комірка, або доки не виявиться що досліджені всі комірки таблиці.

Подвійне хешування краще лінійного та квадратичного дослідження в сенсі кількості $\Theta(m^2)$ послідовностей досліджень. Це пов'язано з тим, що кожна можлива пара $(h_1(k), h_2(k))$ дає свою, відмінну від інших послідовність досліджень. У результаті продуктивність подвійного хешування достатньо близька до продуктивності „ідеальної" схеми рівномірного хешування.

Аналіз відкритої адресації, як і аналіз методу ланцюгів, виконується із використанням коефіцієнту заповнення $\alpha = n/m$ хеш-таблиці при n та m, які прямують до нескінченності. Зрозуміло, при використанні відкритої адресації $\alpha \le 1$, адже не може бути більше одного елементу на комірку таблиці ($n \le m$).

Будемо вважати, що використовуємо рівномірне хешування. За такою ідеалізованою схемою послідовність досліджень $\left[h(k,0), h(k,1), \ldots, h(k,m-1)\right]$, яка використовується для вставки та пошуку кожного ключа k, з рівною ймовірністю є однією з можливих перестановок множини $\{0,1,\ldots m-1\}$. З кожним конкретним ключем пов'язана одна послідовність досліджень, тому під час розгляду розподілу ймовірностей ключем та хеш-функцій всі послідовності досліджень виявляються рівноймовірними.

Теорема 9.3. Математичне сподівання кількості досліджень під час невдалого пошуку в хеш-таблиці з відкритою адресацією та коефіцієнтом заповнення $\alpha < 1$ при припущенні рівномірного хешування не більше величини $1/(1-\alpha)$.

Доведення. При невдалому пошуку кожна послідовність досліджень завершується на порожній комірці. Визначимо випадкову величину X рівну кількості досліджень, які виконуються під час невдалого пошуку, та події $A_i \ (i = 1,2,\ldots)$, які полягають в тому, що було виконано i-е дослідження, і воно прийшлось на заповнену комірку. Тоді подія $\{X \ge 1\}$ є перетином подій $A_1 \cap A_2 \cap \ldots \cap A_{i-1}$. Обмежимо ймовірність $\Pr\{X \ge 1\}$ шляхом обмеження ймовірності $\Pr\{A_1 \cap A_2 \cap \ldots \cap A_{i-1}\}$. За узагальненою формулою умовних ймовірностей маємо:

$$\Pr\{A_1 \cap A_2 \cap \ldots \cap A_{i-1}\} = \Pr\{A_1\} \cdot \Pr\{A_2 | A_1\} \cdot \Pr\{A_1 | A_1 \cap A_2\} \ldots \Pr\{A_{i-1} | A_1 \cap A_2 \cap \ldots \cap A_{i-2}\}$$

Оскільки всього є n елементів та m комірок, $\Pr\{A_1\} = n/m$. Ймовірність того, що буде виконане j-е дослідження ($j > 1$) та воно буде проведене над заповненою коміркою (при цьому перші $j-1$ досліджень проведені над заповненими комірками), дорівнює $(n-j+1)/(m-j+1)$. Ця ймовірність визначається наступним чином: ми повинні перевірити один з $n-(j-1)$ елементів

що лишились, а всього недосліджених на цей час комірок лишається $m-(j-1)$. За припущенням рівномірного хешування ймовірність дорівнює відношенню цих величини. Скориставшись тим фактом, що з $n<m$ для всіх $0\le j<m$ слідує співвідношення $(n-j)/(m-j)\le n/m$, для всіх $1\le i<n$ отримуємо:

$$\Pr\{X\ge i\}=\frac{n}{m}\cdot\frac{n-1}{m-1}\cdot\frac{n-2}{m-2}\cdots\frac{n-i+2}{m-i+2}\le\left(\frac{n}{m}\right)^{i=1}=\alpha^{i-1}$$

Звідси можна обмежити математичне сподівання кількості досліджень:

$$E[X]=\sum_{i=1}^{\infty}\Pr\{X\ge 1\}\le\sum_{i=1}^{\infty}\alpha^{i-1}=\sum_{i=0}^{\infty}\alpha^{i}=\frac{1}{1-\alpha}.$$

■

Отримана межа $1+\alpha+\alpha^2+\alpha^3+\ldots$ має інтуїтивну інтерпретацію. Одне дослідження виконується завжди. З ймовірністю, яка близька до α, перше дослідження проводиться над заповненою коміркою, а тому потрібне ще одне дослідження. З ймовірністю, яка близька до α^2, дві перші комірки виявляються заповненими, а тому потрібне ще одне дослідження, і т.д.

Якщо α – стала, то теорема 9.3 передбачає, що невдалий пошук виконується за час O(1). Наприклад, якщо хеш-таблиця заповнена наполовину, то середня кількість досліджень при невдалому пошуку буде не більша за $1/(1-0{,}5)=2$. При наповненості хеш-таблиці на 90% середня кількість досліджень буде не більша за $1/(1-0{,}9)=10$.

Теорема 9.3 також дає безпосередню оцінку продуктивності процедури `HashInsert()`, яка полягає в тому, що середня кількість досліджень при вставці елементу в хеш-таблицю з урахуванням рівномірного хешування не більша за $1/(1-\alpha)$.

Теорема 9.4. Математичне сподівання кількості досліджень під час вдалого пошуку в хеш-таблиці з відкритою адресацією та коефіцієнтом заповнення $\alpha<1$ при припущенні рівномірного хешування не більше величини

$$\frac{1}{\alpha}\ln\frac{1}{1-\alpha}.$$

Доведення. Пошук ключа k виконується в тій самій послідовності досліджень, що й його вставка. Якщо k був $(i+1)$-м ключем, який був вставлений в хеш-таблицю, то математичне сподівання кількості досліджень при пошуку k буде не більшим за $1/\left(1-i/m\right)=m/\left(m-i\right)$. Усереднення по всім n ключам в хеш-таблиці дає нам середню кількість досліджень при вдалому пошуку:

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i}=\frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i}=\frac{1}{\alpha}\left(\frac{1}{m}+\frac{1}{m-1}+\frac{1}{m-2}+\ldots+\frac{1}{m-n+1}\right)=\frac{1}{\alpha}\sum_{k=m-n+1}^{m}\frac{1}{k}\leq$$

$$\leq\frac{1}{\alpha}\int_{m-n}^{m}(1/x)dx=\frac{1}{\alpha}\ln\frac{m}{m-n}=\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$$

Якщо хеш-таблиця заповнена наполовину, очікувана кількість досліджень при вдалому пошуку не більша за 1,387; при наповненості в 90% ця величина не перевищує 2,559.