НЕЛІНІЙНІ СТРУКТУРИ ДАНИХ
ХЕШ-ТАБЛИЦІ

Лекція 7

Дисципліна «Алгоритми та структури даних»

Обмеження лінійних структур та бінарних дерев

Лінійні структури зберігають елементи послідовно, тому основні операції пошуку (Search), вставки (Insert) та видалення (Delete) мають такі характеристики:

Структура	Пошук (Search)	Вставка (Insert)	Видалення (Delete)
Невідсортований масив/список	O(N)	O(1) / O(N)	O(N)
Відсортований масив	O(logN) (Бінарний пошук)	O(N) (Потрібен зсув елементів)	O(N)

Навіть у найкращому випадку (для несортованого масиву) пошук елемента вимагає перегляду, в середньому, N/2 елементів, що дає лінійну складність O(N). При великій кількості даних це дуже повільно

Дерева були створені, щоб подолати лінійну складність пошуку. У збалансованих деревах операції Search, Insert та Delete виконуються логарифмічний час:

Складність: O(log N). Це значне покращення.
Наприклад, для N= 1,000,000 елементів, O(log N) ≈20 операцій, тоді як O(N) = 1,000,000 операцій

Хоча логарифм росте дуже повільно, він все ж залежить від розміру вхідних даних N

Мета хеш-таблиць: досягти *ідеальної* швидкості операцій (Search, Insert, Delete) за **константний час** O(1) в середньому випадку, незалежно від загальної кількості елементів N
Це досягається шляхом **відображення ключа безпосередньо в адресу пам'яті** за допомогою математичної функції (хеш-функції), замість того, щоб шукати його через порівняння або переходи по дереву

Таблиці прямої адресації (Direct Address Tables)

Таблиця прямої адресації (T) - це специфічна структура даних, яка реалізує словник (dictionary) або асоціативний масив, де **ключ елемента (key)** використовується **безпосередньо** як **індекс** у масиві

Розглянемо невелику інформаційну систему для університету, яка зберігає дані про кафедри

Припускаємо, що:
1) Університет має максимум 100 кафедр
2) ID кафедри є унікальним цілим числом від 0 до 99

1. Налаштування:

Множина ключів (U): {0, 1, …, 99}

Розмір таблиці (M): 100.

Таблиця (T): Масив розміром 100, де кожен індекс відповідає ID кафедри

2. Виконання Операцій O(1):

Припустимо, є такі ключі (ID) та значення (Назва кафедри):

k_1 = 15 (Кафедра комп'ютерних наук)
k_2 = 7 (Кафедра математики)
k_3 = 88 (Кафедра фізики)

Операція	Ключ (k)	Дія	Результат
Вставка	15	T[15] ← Кафедра комп'ютерних наук	Миттєвий запис
Вставка	7	T[7] ← Кафедра математики	Миттєвий запис
Пошук	15	Повертає T[15]	Отримано: Кафедра комп'ютерних наук
Видалення	88	T[88] ← NIL	Миттєве видалення

Якщо необхідно зберігати ідентифікатори студентів (ID), які можуть мати 10-значний номер, то:

множина ключів: U = {1000000000, …, 9999999999} (Приблизно 10^{10} ключів)

розмір таблиці (M): Потрібно створити масив розміром 10^{10} елементів

Якщо один елемент (запис про студента) займає 100 байт, загальний обсяг пам'яті буде близько **1 ТБ**

Хоча O(1) збережеться, пам'ять буде витрачена неефективно, оскільки університет має лише, наприклад, 10 000 студентів, і 99.99% комірок залишаться порожніми.

Для подолання цього **неефективного використання пам'яті** (Space Inefficiency) переходимо до **хеш-таблиць**, які використовують **хеш-функцію** для відображення величезного ключа (k=1000000000) на маленький індекс (i=15) у доступному масиві

Визначення хеш-таблиці та асоціативного масиву

Асоціативний масив (словник, Map) - абстрактний тип даних, який зберігає колекцію пар *(ключ, значення)*. Кожен унікальний ключ асоціюється з одним значенням

Основні операції:

- додати пару (ключ k, значення v);
- знайти та повернути значення, асоційоване з ключем k;
- видалити пару з ключем k.

Хеш-таблиця (Hash Table) - структура даних, яка реалізує асоціативний масив, використовуючи масив (таблицю) та **хеш-функцію** для відображення ключів на індекси цього масиву.

Таблиця (T**) -** масив, який зберігає самі дані

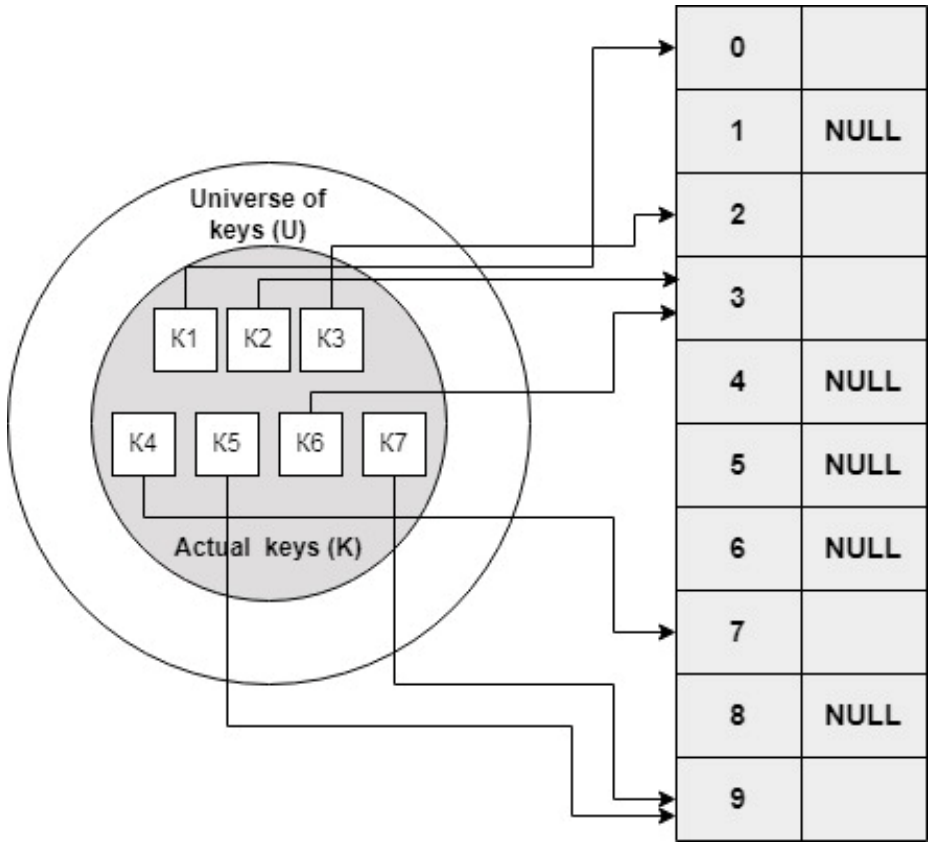
Хеш-функція (h**) -** функція, яка приймає ключ ***k*** і повертає індекс ***i*** у таблиці **T**

тобто $i = h(k)$

Принцип роботи: Замість використання ключа безпосередньо як індексу (як у прямій адресації), хеш-таблиця використовує хеш-функцію для стиснення великого діапазону ключів до меншого діапазону індексів таблиці.

Хеш-таблиці. Колізії

Перетворення значень ключа до виду, придатного для використання в хеш-таблиці, називається **хешуванням** і описується особливими математичними **хеш-функціями**. Хешовані значення ключів часто схожі, тому хороші хеш-функції розподіляють їх таким чином, щоб вони розташовувалися в різних позиціях в таблиці



Кожен ключ з множини ключів K зіставляється з індексом в таблиці, що генерується за допомогою хеш-функції
Ключі K2 і K6, а також K5 і K7 демонструють виникнення колізії (конфлікту). Рядки таблиці з індексами 1, 4-6 і 8 є порожніми і готовими до розміщення нових записів
Якщо в хеш-таблиці розміщується досить багато значень, рано чи пізно все одно знайдуться два ключа, які виявляться пов'язаними з однією і тією ж позицією в таблиці. Така ситуація називається **колізією**.

Коефіцієнт заповнення хеш-таблиці відображає те, наскільки багато в ній записів з даними у порівнянні з її об'ємом. Цей показник впливає на ймовірність виникнення колізій. Більше шансів, що вони будуть мати місце при додаванні ключа в таблицю, заповнену на 95%, ніж на 10%

Хеш-функція. Властивості

Хеш-функція являє собою математичну формулу, яка при застосуванні до ключа обчислює ціле число, яке може бути використане в якості індексу для хештаблиці

Властивості хеш-функції:

Бюджетність. Витрати на обчислення хеш-функції повинні бути невеликими, так щоб час хешування був меншим, ніж використання інших підходів до представлення даних. Наприклад, якщо алгоритм бінарного пошуку може шукати елемент у відсортованій таблиці з N елементів за час O(log2N), то хеш-функція повинна виконуватися ще швидше

Детермінізм. Для одного й того ж ключа хеш-функція повинна повертати один і той самий індекс. Цей критерій виключає хеш-функції, які залежать від зовнішніх параметрів середовища виконання (наприклад, часу доби) і від адреси пам'яті хешованого об'єкта (так як адреса об'єкта може змінюватися в процесі обробки)

Однорідність. Гарна хеш-функція повинна генерувати індекс в таблиці по ключу якомога більш рівномірно по всій своїй області значень. Це означає, що ймовірність появи кожного значення хеш-функції в області значень повинна бути однаковою. Властивість однорідності дозволяє звести до мінімуму число колізій

Обчислення залишку (метод ділення)

Це найпростіший метод хешування цілого ключа ***k***. Цей метод ділить націло ключ ***k*** на число **M** (***k*** > **M**), а потім використовує отриманий в результаті залишок. В цьому випадку, хеш-функція може бути задана у вигляді

$$h(k) = k \bmod M$$

k – ключ, ціле число;

M – розмір хеш-таблиці

Індекс є залишком від ділення ключа на розмір таблиці

Як правило, краще всього вибирати M простим числом, тому що просте число збільшує ймовірність того, що генерація хеш-значення по ключу відбуватиметься рівномірно по всій області значень функції

Приклад: хешування ID товарів (метод ділення)

Завдання. Невеликий склад використовує 4-значні ідентифікатори (ID) для своїх товарів (від 1000 до 9999). Оскільки активний запас товару рідко перевищує 15 одиниць, керівництво вирішило використати невелику хеш-таблицю для швидкого пошуку інформації про товар

Мета. Здійснити відображення унікальних 4-значних ID товарів на індекси таблиці розміром 13

Таблиця (T**) -** масив розміром M=13. Валідні індекси: 0, 1, …, 12

Розмір таблиці (M**) -** 13. (Це **просте число**, що є практикою для методу ділення, оскільки сприяє кращому розподілу ключів)

Хеш-функція - використовуємо метод ділення (обчислення залишку):

$$h(k) = k \bmod 13$$

Обчислення індексу для кожного ключа за формулою $h(k) = k \mod 13$

Вхідні Ключі (ID Товарів)
Припустимо, потрібно вставити інформацію про такі товари:
k1 = 4371 (Монітор 27")
k2 = 8105 (Дротова клавіатура)
k3 = 6930 (Оптична миша)
k4 = 1001 (USB-кабель)

Ключ (k)	Обчислення: k mod 13	Індекс (h(k))
4371	4371 div 13 = 336 (залишок 3)	**3**
8105	8105 div 13 = 623 (залишок 6)	**6**
6930	6930 div 13 = 533 (залишок 1)	**1**
1001	1001 div 13 = 77 (залишок 0)	**0**

Індекс	Збережені Дані (Товар)
0	ID 1001 (USB-кабель)
1	ID 6930 (Оптична миша)
2	NULL
3	ID 4371 (Монітор 27")
4	NULL
5	NULL
6	ID 8105 (Дротова клавіатура)
7	NULL
...	...
12	NULL

Виконання операції пошуку O(1)

Припустимо, менеджеру потрібно швидко знайти інформацію про товар з ID **4371**.
Обчислення: h(4371) = 4371 mod 13 = 3
Пошук: Менеджер прямо звертається до слота T[3]
Результат: Отримує інформацію: "Монітор 27"".
Висновок: Завдяки хеш-функції, система миттєво (за O(1)) перетворює великий, довільний ID на конкретний індекс пам'яті, що значно прискорює пошук записів

Мультиплікативний метод (метод множення)

Метод множення часто використовується у внутрішніх системах, таких як управління кеш-пам'яттю, де бажано мати розмір таблиці, який є степенем двійки, для оптимізації швидкості

Алгоритм мультиплікативного методу :

Крок 1: Вибрати константу A таку, що $0 < A < 1$

Крок 2: Помножити ключ $\boldsymbol{k}$ на A.

Крок 3: Взяти дробову частину добутку кроку 2.

Крок 4: Помножити результат кроку 3 на розмір хеш-таблиці M.

Крок 5: Взяти цілу частину результату Крок 4.

Отже, хеш-функцію можна визначити як:

$$\boldsymbol{h(k) = \lfloor M * (k * A)(mod\ 1)) \rfloor}$$

$\boldsymbol{k}$- ключ (ціле число); **M** - розмір хеш-таблиці; **A** - константа, де $\mathbf{0 < A < 1}$

Найголовніша перевага цього методу полягає в тому, що він працює практично з будь-яким значенням A. Хоча оптимальний вибір значення A все ж існує. Дональд Кнут в своїй знаменитій багатотомній праці, присвяченій алгоритмам, припустив, що найкращим вибором для A є вибір числа $\varphi = 0.6180339887$ ("золотий" перетин)

Приклад. Хешування адрес пам'яті (метод множення)

Завдання. Розглянемо систему кешування, де необхідно швидко відобразити 16-бітні адреси пам'яті на невелику кеш-таблицю розміром 64

Мета: Відобразити великий діапазон 16-бітних адрес (k) на індекси кеш-таблиці розміром 64 (M=64)

Розмір таблиці (M**) -** 64 (2^6).

Хеш-функція – використовуємо метод множення

Константа: A = 0.6180339887

Формула: $\boldsymbol{h(k) = \lfloor M * (k * A)(mod\ 1))\rfloor}$

Обчислення індексу для кожного ключа за формулою $h(k) = \lfloor M * (k * A)(mod\ 1)) \rfloor$

Вхідні ключі (адреси ам'яті)

Припустимо, хешуємо такі 16-бітні адреси (у десятковій системі):

k1 = 12345

k2 = 20000

Висновок: адреса k1=12345 буде збережена у слоті T[61], а адреса k2=20000 - у слоті T[43]

Крок	Дія	Обчислення для k1 = 12345	Обчислення для k2 = 20000
1	Множення на A	12345 *0.61803 = 7630.95758	20000 * 0.61803 = 12360.6797
2.	Взяття дробової частини (***mod 1***)	7630.95758 mod 1 = 0.95758	12360.6797 mod 1 = 0.6797
3.	Множення на M=64	64 * 0.95758 = 61.285	64 * 0.6797= 43.5008
4.	Взяття цілої частини	61.285 = 61	43.5008 = 43
Результат	Індекс ***h(k)***	**61**	**43**

Переваги методу множення у цій задачі

Незалежність від M: Метод множення менш чутливий до вибору M. Навіть якщо M є степенем двійки, вибір константи A (особливо золотого перетину) забезпечує відмінне рівномірне хешування, оскільки дробова частина ***k*A*** залежить від усіх ***k***

Оптимізація швидкості: У системах, де M= 2^p, вся операція може бути виконана надзвичайно швидко за допомогою зсувів та бітових операцій, що критично важливо для кеш-пам'яті процесора

Поліноміальний хеш (Polynomial Rolling Hash)

Використання - ідеально підходить для **рядкових ключів** (strings)

Принцип - розглядає рядок як число в системі числення з основою p (де p - невелике просте число, наприклад, 31 або 37), а потім обчислює це число по модулю M

Функція:

$$h(S) = \left(\sum_{i=0}^{L-1} S[i] \cdot p^i \right) \pmod{M}$$

S[i] **-** ASCII-значення i-го символу рядка

L **-** довжина рядка

p **-** основа (просте число)

M **-** розмір хеш-таблиці (модуль); **це значення гарантує, що кінцевий хеш-індекс потрапить у діапазон** [0, M-1]

Перевага: Дуже добре розподіляє рядкові ключі та є основою для багатьох ефективних алгоритмів обробки рядків

Приклад. Поліноміальний хеш для рядків

Завдання: Обчислити хеш-індекс для рядка **"CAT"**

Параметр	Значення	Призначення
Ключ (S)	"CAT"	Рядок для хешування
Розмір таблиці (M)	101	Модуль (залишок від ділення)
Основа (p)	31	База поліному

Функція: $$h(S) = \left(\sum_{i=0}^{L-1} S[i] \cdot p^i \right) \pmod{M}$$

Числові значення символів (ASCII-коди)

Спочатку перетворимо символи на їхні числові значення ***S[i]***:

Символ	Індекс (i)	ASCII-код (S[i])
C	0	67
A	1	65
T	2	84

Покрокове обчислення поліному

Обчислюються значення поліному для кожного символу, починаючи з i=0:

Символ	Обчислення (S[i]*p^i)	Результат
C	67 * 31^0 = 67 * 1	67
A	65 * 31^1 = 65 * 31	2015
T	84 * 31^2 = 84 * 961$	80724
Сума		**82806**

Обчислення кінцевого хеш-індексу

Тепер застосовуємо операцію **mod** на суму поліному:

h("CAT") = 82806 mod 101

Для обчислення залишку ділимо суму на модуль:

82806 div 101 = 820 (з залишком) 86

Таким чином:

h("CAT") = 86

Висновок: Рядок "CAT" буде збережено у слоті хеш-таблиці з індексом **86**

Розв'язання колізій

Яким би не був вибір хеш-функції, колізії будуть відбуватися кожного разу, коли хеш-функція ***h(k)*** повертає однаковий індекс для двох різних ключів ***k1 ≠ k2***

Механізми вирішення колізій гарантують, що обидва ключі можуть бути збережені та знайдені

Два з найбільш популярних методів розв'язання колізій у хешуванні:

1. Метод відкритої адресації
2. Метод ланцюжків (пряме зв'язування)

Метод ланцюжків (Separate Chaining)

Принцип роботи:

Замість того, щоб зберігати безпосередньо дані в кожному слоті таблиці T[i], кожен слот стає **покажчиком на зв'язаний список (ланцюжок)**

Якщо ключ k хешується до індексу i (тобто h(k)=i), то ключ k та його значення додаються в кінець зв'язаного списку, розташованого у слоті T[i]

Якщо слот T[i] порожній, створюється новий список із цим елементом

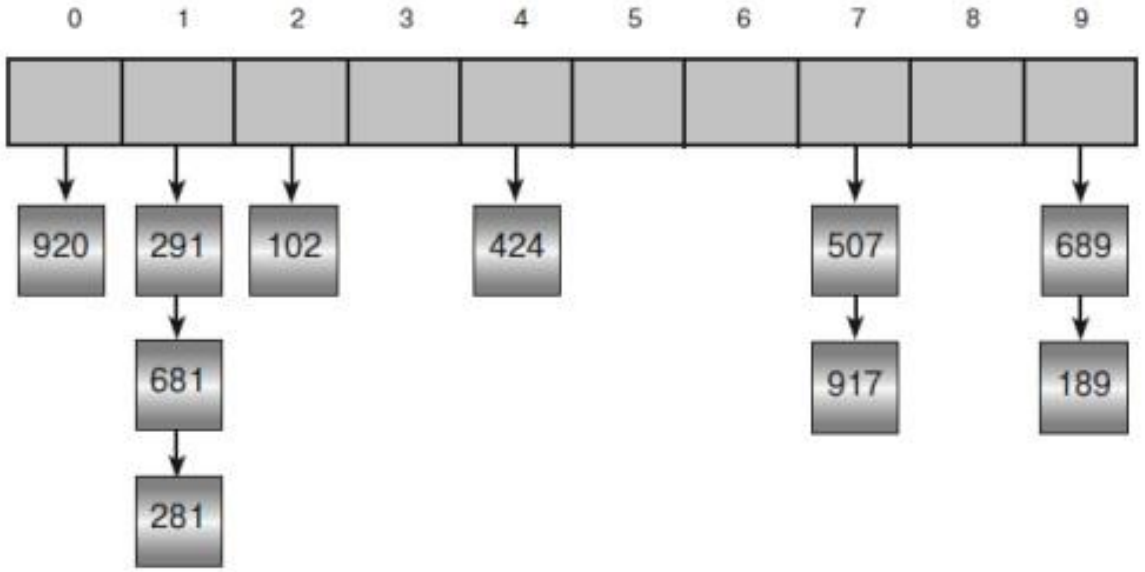
Операції

Вставка: обчислити h(k) і додати елемент на початок або кінець списку в T[h(k)] (O(1) у середньому)

Пошук: обчислити h(k) і пройти по зв'язаному списку в T[h(k)] до знаходження ключа. Складність залежить від довжини списку

Метод ланцюжків

У хеш-таблиці з прямим зв'язуванням значення ключів зберігаються в спеціальних наборах записів - **блоки переповнення**. Кожен з елементів самої хеш-таблиці є фактично **дескриптором (покажчиком) зв'язного списку**, в якому знаходяться прив'язані до блоку переповнення елементи. У самій таблиці в позиції, що задається хеш-функцією, зберігається тільки значення покажчика адреси "прив'язаного" до цього місця блоку (так званий метод із зовнішніми ланцюжками). Тобто, хеш-таблиця є простою адресною таблицею, і **записів** з інформаційними полями (даними) **не містить**



Зв’язані списки в оперативній пам'яті створюються у динамічній області пам'яті, коли для кожного елемента виділяється пам'ять і для кожної хеш-адреси створюється свій блок переповнення (ланцюжків)

Приклад: хешування ID товарів. Колізія

Вхідні ключі (ID товарів)

Припустимо, нам потрібно вставити інформацію про такі товари:

k1 = 4371 (Монітор 27")

k2 = 8105 (Дротова клавіатура)

k3 = 6930 (Оптична миша)

k4 = 4384 (Кабель HDMI) **- новий ID для демонстрації колізії**

Ключ (k)	Обчислення: k mod 13	Індекс (h(k))
4371	4371 div 13 = 336 (залишок 3)	**3**
8105	8105 div 13 = 623 (залишок 6)	**6**
6930	6930 div 13 = 533 (залишок 1)	**1**
4384	4384 div 13 = 337 (залишок 3) КОЛІЗІЯ з ID 4371	**3**

Хеш-таблиця після використання методу ланцюжків

Індекс	Збережені Дані (Товар)
0	ID 1001 (USB-кабель)
1	ID 6930 (Оптична миша)
2	NULL
3	ID 4371 (Монітор 27") → ID 4384 (Кабель HDMI) (**Ланцюжок**)
4	NULL
5	NULL
6	ID 8105 (Дротова клавіатура)
7	NULL
...	...
12	NULL

Виконання операції пошуку з колізією

Припустимо, менеджеру потрібно швидко знайти інформацію про товар з ID **4384**
Обчислення: h(4384) = 4384 mod 13 = 3
Пошук: Менеджер звертається до слота T[3]
Вирішення колізії: Оскільки слот T[3] містить ланцюжок, менеджер повинен перебрати елементи ланцюжка, порівнюючи ключі:

Перший елемент: Ключ 4371 ≠ 4384
Другий елемент: Ключ 4384 = 4384
ЗНАЙДЕНО!

Результат: менеджер отримує інформацію: “Кабель HDMI"

Метод відкритої адресації

На відміну від методу ланцюжків, відкрита адресація зберігає **всі елементи безпосередньо у слотах самої хеш-таблиці**. У таблиці не використовуються зв'язані списки

Принцип роботи:

Якщо слот T[i] зайнятий (колізія), система послідовно "зондує" (пробиває, шукає) наступні альтернативні слоти, поки не знайде вільний.

Функція зондування:

$$h(k, i) = (h'(k) + f(i)) \bmod M$$

h' (k) **-** основна хеш-функція (наприклад, метод ділення**)**

i **-** номер спроби (зондекса), починаючи з i=0

f(i) **-** функція, що визначає послідовність зондування

Операції:

Вставка - послідовно перевіряти h(k, 0), h(k, 1), h(k, 2), … до знаходження вільного слота

Пошук - послідовно перевіряти ту ж саму послідовність слотів, поки не буде знайдено ключ k або порожній слот (що означає, що ключа немає)