Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгоритми та складність

Лабораторна робота №1

“Реалізація ідеального хешування”

Виконала студентка 2-го курсу

Групи ІПС-21

Хвала Владислава Андріївна

2025

**Завдання**:

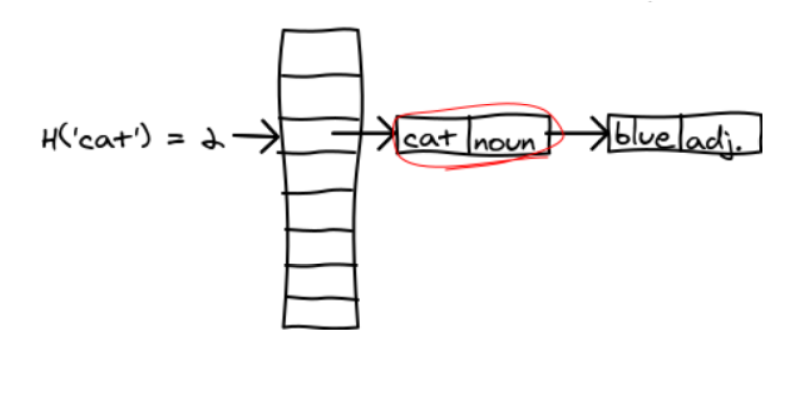
Реалізувати ідеальне хешування.

**Тип даних Т1 (рядки)**:

String

**Теорія**:

Вигляд звичайної хеш-таблиці:

****

Як бачимо, вона краща за звичайний словник тим, що дозволяє зберігати додаткову інформацію про слова.

Ідеальна хеш-функція – хеш функція, яка перетворює завчасно відому статичну множину ключів в діапазоні цілих чисел [0,n-1] без колізій (випадків, коли 2 рядки мають однакове хеш-значення), тобто один ключ відповідає лише одному унікальному значенню.

Вона має такі властивості:

• Ο(1) звертань до пам’яті в найгіршому випадку.

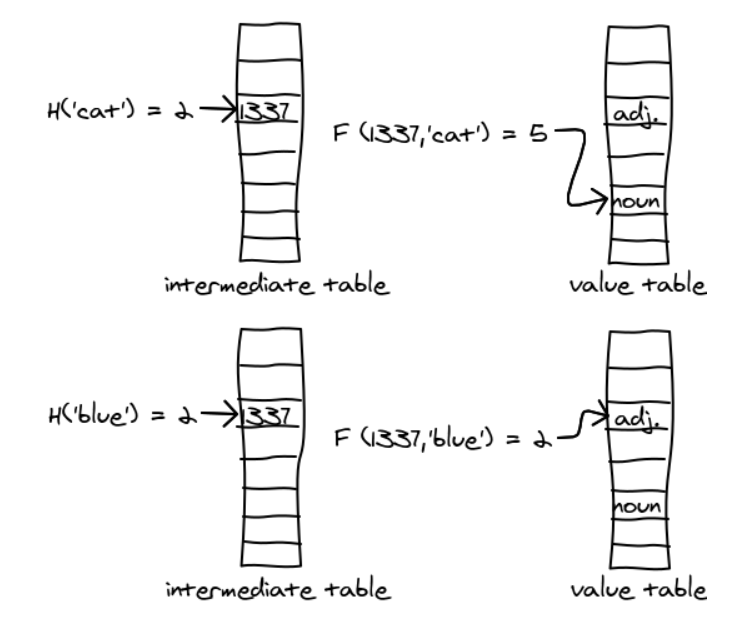
• Множина ключів статична – не змінюється після збереження в таблицю.

• Перший рівень хешування: n ключів хешуються в m комірок за допомогою універсальної хеш-функції h.

• Другий рівень хешування: для кожної комірки своя вторинна хеш-таблиця зі своєю універсальною хеш-функцією, вибраною так, щоб уникнути колізій; її розмір – квадрат кількості ключів, захешованих в комірку.

• Очікувана загальна пам’ять під таку структуру O(n).

Таблиця ідеального хешування:



Наші слова "blue" і "cat" обоє хешуються в ту саму позицію, згідно з першим рівнем.

Однак другий рівень кладе їх в різні комірки.

Тож пошук елементів здійснюється методом спроб і помилок, що займає лінійний час.

**Алгоритм**:

Для зручності позначимо рівні як H(key) - визначає позицію в проміжному масиві G і  F(d, key) - використовує додаткову інформацію з G для знаходження унікальної позиції для ключа.

1. розміщуємо ключі в кошиках (так званих “buckets”) відповідно до першої хеш-функції, H.
2. обробляємо кошики починаючи з найбільших і намагаємося розмістити всі ключі, що вони містять, у порожні місця в таблиці значень за допомогою F(d=1, key). Якщо це не вдається, ми продовжуємо пробувати з більшими значеннями d. Може здатися, що це займе багато часу, але насправді це не так. Оскільки спершу ми намагаємося знайти значення d для кошиків з найбільшими кількостями елементів, ймовірно, вони займуть порожні місця. Коли ми дійдемо до кошиків, що містять лише один елемент, ми просто можемо розмістити його у наступному вільному місці.

**Складність**:

Алгоритм має константний час O(n), а пошук працює за час Ο(1) в найгіршому випадку.

**Мова програмування**:

Python

**Модулі програми:**

Обчислює унікальну хеш-функцію для заданого рядка. Кожне значення цілої величини d дає різне хеш-значення.

def hash(d, str):

Обчислює мінімальну ідеальну хеш-таблицю, використовуючи наданий початковий словник Python. Вона повертає кортеж (G, V). G і V є масивами. G містить проміжну таблицю значень, необхідних для обчислення індексу значення в V. V містить значення зі словника.

def CreateMinimalPerfectHash(dict):

Шукає значення в хеш-таблиці, визначеній за допомогою G та V.

def PerfectHashLookup(G, V, key):

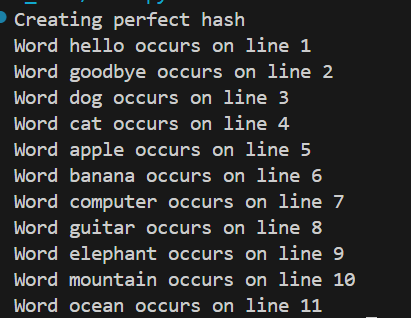
**Інтерфейс користувача**

Вхідні дані вводяться програмно (в словник) і виводяться в консоль.

**Тестові приклади**

Маємо словник вхідних слів

TEST\_WORDS = ['hello', 'goodbye', 'dog', 'cat', 'apple', 'banana', 'computer', 'guitar', 'elephant', 'mountain', 'ocean']



Спочатку ми створюємо **словник** (dict), де кожному слову в масиві присвоюється унікальний індекс, що збільшується від 1 до n (де n — кількість слів). У вашому випадку для масиву ['hello', 'goodbye', 'dog', 'cat', 'apple', 'banana', 'computer', 'guitar', 'elephant', 'mountain', 'ocean'], словник буде виглядати так:

dict = {

'hello': 1,

'goodbye': 2,

'dog': 3,

'cat': 4,

'apple': 5,

'banana': 6,

'computer': 7,

'guitar': 8,

'elephant': 9,

'mountain': 10,

'ocean': 11

}

Кожне слово з словника розподіляється по **відрах** (buckets), які обчислюються за допомогою хеш-функції.

Ми сортуємо відра за кількістю елементів.

Далі розміщуємо елементи по таблиці.

Коли всі елементи в відрі успішно розміщені, ми заносимо їх індекси в масив **G** та значення в масив **V**.

Для кожного слова з масиву TEST\_WORDS, ми використовуємо функцію PerfectHashLookup, щоб знайти його індекс у таблиці.

**Висновки:**

Ідеальне хешування використовується рідко, проте його алгоритм не є складним та важким для програмування. Його складність лінійна.

**Література:**

* Лекція з предмету «Алгоритми та складність #1»
* <https://stevehanov.ca/blog/?id=119>
* https://en.wikipedia.org/wiki/Perfect\_hash\_function