МНІСТЕРСТВО НАУКИ ТА ОСВІТИ УКРАЇНИ
Національний Авіаційний Університет
Факультет комп'ютерних наук та технологій
Кафедра прикладної математики


# "Алгоритми та структури даних"

Лабораторна робота №2.2

16 листопада 2023 р.

Виконав:
студент №14 групи ПМ 2516 НАУ
Владислав Реган Володимирович

Прийняв:
доцент, к.т.н.
Чолишкіна Ольга Генадіївна

Київ 2023

# Зміст

# 1. Тема і мета роботи

**Тема:** Хеш-функції. Хеш-таблиці.
**Мета:** Ознайомитися з основними алгоритмами хешування даних і обробки хеш-таблиць.

# 2. Постановка задачі

1. Вивчити основні алгоритми побудови хеш-функцій.
2. Вивчити методи побудови хеш-таблиць і основні алгоритми пробірування.
3. Реалізувати програмно хеш-таблицю з використанням методу, зазначеним у варіанті. Організувати обробку хеш-таблиці відповідно до запиту користувача.
4. Оцінити обчислювальну складність алгоритмів обробки.

Варіант 0 (варіант $= N \mod 7$ де $N = 14$)

Реалізувати програмно хеш-таблицю з використанням методу ланцюжків.

# 3. Теоретичні відомості

Вміст цієї секції взято з [1].

## 3.1. Хеш таблиці

### 3.1.1. Що таке хешування?

Хешування - це техніка, яка використовується для зберігання та швидкого пошуку інформації. Використовується для виконання оптимального пошуку і є корисним при реалізації таблиць символів.

### 3.1.2. Звичайні операції над хеш таблицями

Найпоширенішими операціями для хеш-таблиці є:

- `HashSearch` Шукає ключ у хеш-таблиці
- `Hashlnsert` вставляє новий ключ у хеш-таблицю
- `HashDelete` Видаляє ключ з хеш-таблиці
- `DeleteHashTable` Видаляє хеш-таблицю

### 3.1.3. Елементи хешування

Хешування має чотири ключові компоненти:

1. Хеш-таблиця
2. Хеш-функції
3. Колізії
4. Методи вирішення колізій

### 3.1.4. Хеш таблиця

Хеш-таблиця є узагальненням масиву. У масиві ми зберігаємо елемент з ключем $k$ у позиції $k$ масиву. Це означає, що за заданим ключем $k$ ми можемо знайти елемент, просто подивившись у $k$-ту позицію масиву. Це називається прямою адресацією.

Пряма адресація застосовується, коли ми можемо дозволити собі виділити масив з однією позицією для кожного можливого ключа. Але якщо у нас недостатньо місця, щоб виділити місце для кожного можливого ключа, то нам потрібен механізм для обробки цього випадку.

Інший спосіб визначення сценарію: якщо у нас менше позицій і більше можливих ключів, то простої реалізації масиву недостатньо.

У таких випадках одним з варіантів є використання хеш-таблиць. Хеш-таблиця або хеш-карта - це структура даних, яка зберігає ключі та пов'язані з ними значення, а хеш-таблиця використовує хеш-функцію для зіставлення ключів з пов'язаними з ними значеннями. Зазвичай ми використовуємо хеш-таблицю, коли кількість фактично збережених ключів невелика порівняно з кількістю можливих ключів.

### 3.1.5. Хеш функція

Хеш-функція використовується для перетворення ключа в індекс. В ідеалі, хеш-функція повинна ставити у відповідність кожному можливому ключу унікальний індекс слоту, але на практиці цього важко досягти.

Для набору елементів хеш-функція, яка ставить у відповідність кожному елементу унікальний слот, називається ідеальною хеш-функцією. Якщо ми знаємо елементи і колекція ніколи не зміниться, то можна побудувати ідеальну хеш-функцію. На жаль, для довільного набору елементів не існує систематичного способу побудови ідеальної хеш-функції. На щастя, нам не потрібно, щоб хеш-функція була ідеальною, щоб отримати ефективну роботу.

Один із способів завжди мати ідеальну хеш-функцію - збільшити розмір хеш-таблиці так, щоб у ній можна було розмістити всі можливі значення в діапазоні елементів. Це гарантує, що кожен елемент матиме унікальний слот. Хоча це практично для невеликої кількості елементів, це нездійсненно, коли кількість можливих елементів велика. Наприклад, якби елементами були дев'ятизначні номери соціального страхування, цей метод вимагав би майже мільярд слотів. Якщо ми хочемо зберігати дані лише для класу з 25 студентів, ми витрачатимемо величезну кількість пам'яті. Наша мета - створити хеш-функцію, яка мінімізує кількість колізій, легко обчислюється і рівномірно розподіляє елементи в хеш-таблиці. Існує декілька поширених способів розширення методу простого залишку.

### 3.1.6. Коефіцієнт заповнення

Коефіцієнт завантаження $L$ непорожньої хеш-таблиці - це кількість $C$ елементів, що зберігаються в таблиці, поділена на розмір таблиці $S$. Це параметр рішення, який використовується, коли ми хочемо перехешувати або розширити існуючі записи хеш-таблиці. Він також допомагає нам визначити ефективність функції хешування. Це означає, що він показує, чи розподіляє хеш-функція ключі рівномірно, чи ні.

$$L = \frac{C}{S} \tag{3.1}$$

### 3.1.7. Колізії (зіткнення)

Хеш-функції використовуються для зіставлення кожного ключа з окремим адресним простором, але практично неможливо створити таку хеш-функцію, і ця проблема називається колізією. Колізія - це стан, коли два записи зберігаються в одному місці.

### 3.1.8. Методи розв'язання колізій

Процес пошуку альтернативного розташування називається розв'язанням колізій. Незважаючи на те, що хеш-таблиці мають проблеми з колізіями, у багатьох випадках вони є більш ефективними порівняно з іншими структурами даних, такими як дерева пошуку. Існує декілька методів вирішення колізій, найпопулярнішими з яких є прямий ланцюжок і відкрита адресація.

- Прямий ланцюжок: Застосування масиву зв'язаних списків
    - Окремий ланцюг

- Відкрита адресація: Реалізація на основі масивів
  - Лінійне зондування (лінійний пошук)
  - Квадратичне зондування (нелінійний пошук)
  - Подвійне хешування (використання двох хеш-функцій)

### 3.1.9. Роздільний ланцюжок

Розв'язання колізій за допомогою ланцюжків поєднує зв'язане представлення з хеш-таблицею. Коли два або більше записів хешуються в одному і тому ж місці, ці записи складаються в однозв'язний список, який називається ланцюжком.

### 3.1.10. Як хешування набуває складності $O(1)$

З попереднього обговорення виникає питання, як хешування отримує $O(1)$, якщо декілька елементів відображаються в одне і те ж місце... Відповідь на цю проблему проста. Використовуючи коефіцієнт завантаження, ми переконуємося, що кожен блок (наприклад, зв'язаний список в методі окремого ланцюжка) в середньому зберігає максимальну кількість елементів, меншу за коефіцієнт завантаження. Крім того, на практиці цей коефіцієнт завантаження є константою (як правило, 10 або 20). Як наслідок, пошук у 20 елементах або 10 елементах стає постійним. Якщо середня кількість елементів у блоці більша за коефіцієнт завантаження, ми перехешуємо елементи з більшим розміром хеш-таблиці. Слід пам'ятати, що при прийнятті рішення про перехешування ми враховуємо середню заповненість (загальна кількість елементів у хеш-таблиці, поділена на розмір таблиці). Час доступу до таблиці залежить від коефіцієнта завантаження, який в свою чергу залежить від хеш-функції. Це відбувається тому, що хеш-функція розподіляє елементи в хеш-таблиці. З цієї причини ми говоримо, що хеш-таблиця дає в середньому $O(1)$ складності. Крім того, ми зазвичай використовуємо хеш-таблиці у випадках коли пошук є більш складним, ніж операції вставки та видалення.

# 4. Про програму

Програма написана на мові програмування C++ з використанням крос-платформового інструментарію розробки програмного забезпечення Qt та інтегрованого середовища розробки (IDE) QtCreator, яке спрощує розробку GUI-додатків.

## 4.1. Про Qt

Qt (вимовляється як "кьют") - це крос-платформне програмне забезпечення для створення графічних інтерфейсів користувача, а також крос-платформних додатків, які працюють на різних програмних і апаратних платформах, таких як Linux, Windows, macOS, Android або вбудованих системах, з невеликими змінами або без змін в базовій кодовій базі, залишаючись при цьому нативним додатком з нативними можливостями і швидкістю.

Наразі Qt розробляється The Qt Company, публічною компанією, що котирується на біржі, та Qt Project з відкритим вихідним кодом, за участю окремих розробників та організацій, що працюють над розвитком Qt. Qt доступна як під комерційними ліцензіями, так і під ліцензіями з відкритим кодом GPL 2.0, GPL 3.0 та LGPL 3.0.

### Структура

`doublyLinkedList.hpp`

Декларація класу `DoublyLinkedList<typename T>`, вузла `LinkedListNode<typename T>` та його методів.

`ChainHashTable.hpp`
: Декларація класу `ChainHashTable<typename T>`, вузла `ChainHashTableNode<typename T>` та його методів.

`mainwindow.h`
: Декларація наслідування класу `MainWindow` з `QMainWindow`, таблиця `ChainHashTable<QString>`. Включення заголовків `Qt` та `chainHashTable.h` та `doublyLinkedList.hpp`.

`main.cpp`
: Ініціалізація `QApplication` та `MainWindow`.

`mainwindow.cpp`
: Ініціалізація методів `MainWindow`, логіка слотів графічного інтерфейсу користувача.

`mainwindow.ui`
: XML-опис графічного інтерфейсу користувача.

`recources.qrc`
: Ресурс-файл `Qt` з медіа, які необхідні для програми.

`Lab1.pro`
: Pro-файл `QtCreator`.

Так як хеш таблиця реалізована на зв'язних списках, то нам спочатку потрібно імплементувати зв'язні списки на C++:

```
template <class T>
struct LinkedListNode {
    T data;
    LinkedListNode* next;
    LinkedListNode* prev;
    LinkedListNode(T value);
};

template<class T>
class DoublyLinkedList {
public:
    DoublyLinkedList();
    ~DoublyLinkedList();

    LinkedListNode<T>* insertFront(T);
    LinkedListNode<T>* insertBack(T);
    T removeFront();
    T removeBack();
    bool isEmpty();
        remove(LinkedListNode<T>* node);
    void display();
    int length();
    LinkedListNode<T>* search(T);

    LinkedListNode<T>* head;
    LinkedListNode<T>* tail;
    int size;

};
```

`LinkedListNode<T>* insertFront(T)`
: Вставляє значення типу `T` на початок списку.

`LinkedListNode<T>* insertBack(T)`
: Вставляє значення типу `T` в кінець списку.

`T removeFront()`
: Видаляє вузол на початку списку.

`T removeBack()`
: Видаляє вузол в кінці списку.

`bool isEmpty()`
: Повертає істину якщо список пустий.

`void remove(LinkdedListNode<T>* node)`
: Видаляє вузол за адресом `node`, якщо такий є.

`void display()`
: Друкує список в стандартний потік.

`int length()`
: Повертає довжину списку.

`LinkedListNode<T>* search(T)`
: Шукає вузол за типом `T`.

Отримавши зв'язні списки, на їх основі можна імплементувати хеш-таблицю:

```
template<class T>
struct ChainHashTableNode {
        DoublyLinkedList<T> list;
};

template<class T>
class ChainHashTable {
        public:
                void insert(T value);
                LinkedListNode<T>* search(T value);
                void remove(T value);
                int size(void);
                int count(void);

        ChainHashTable(int desiredSize, double loadFact, int (*hashFunc)(T, int));

     private:
        int tableSize;
                int tableCount;
        double loadFactor;
                int (*hashFunction)(T vlaue, int size);
                ChainHashTableNode<T>** table;

                void rehash();
};
```

`void insert(T value)`
: Хешує і вставляє в таблицю елемент типу `T`.

`void remove(T value)`
: Видаляє елемент з таблиці.

`int size(void)`
: Повертає розмір таблиці.

`int count(void)`
: Повертає кількість елементів у таблиці.

`ChainHashTable(int desiredSize, double loadFact, int (*hashFunc)(T, int))`
: Створює таблицю розміром `size`, з коефіцієнтом перевантаження `loadFact` і хеш функцією `hashFunc` яка хешує змінні тиипу `T`.

### Опції

Програма ні в якій мірі не реагує на опції командного рядка.

# 5. Тестування

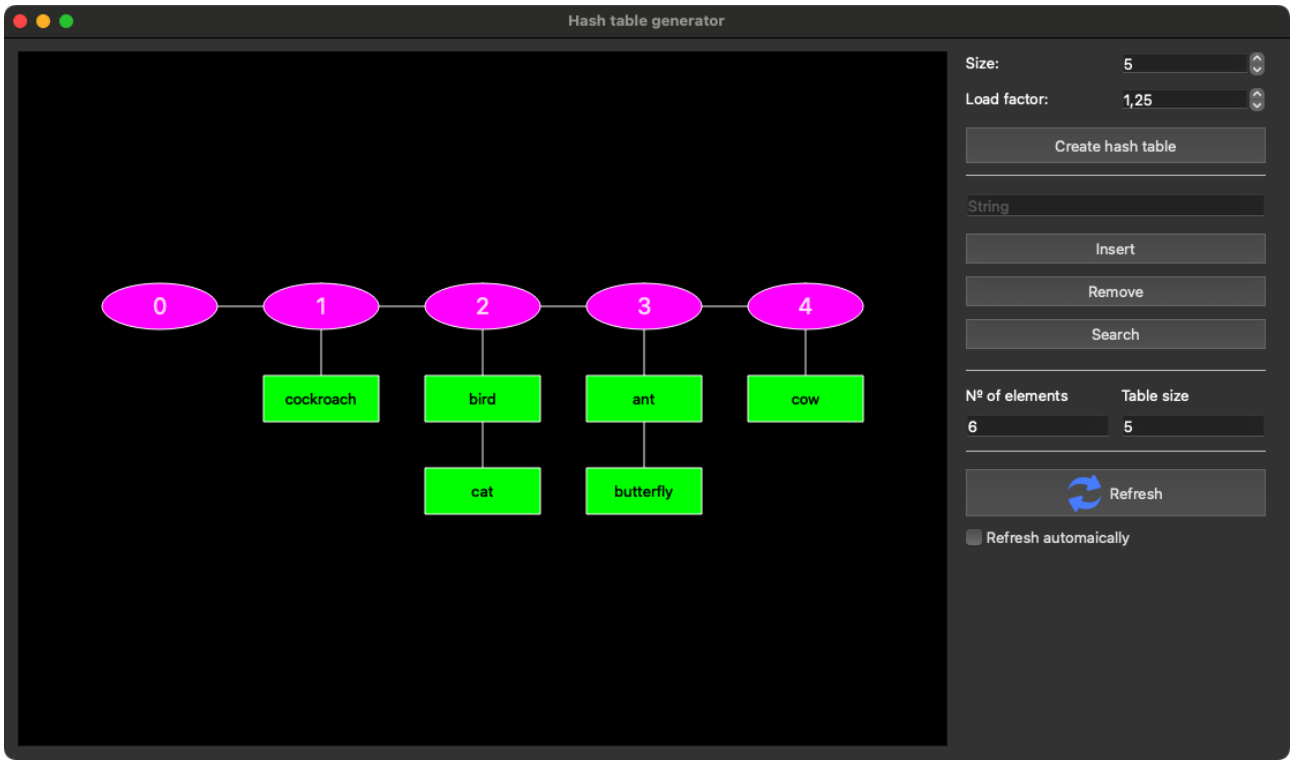


Рис. 1. Запуск програми

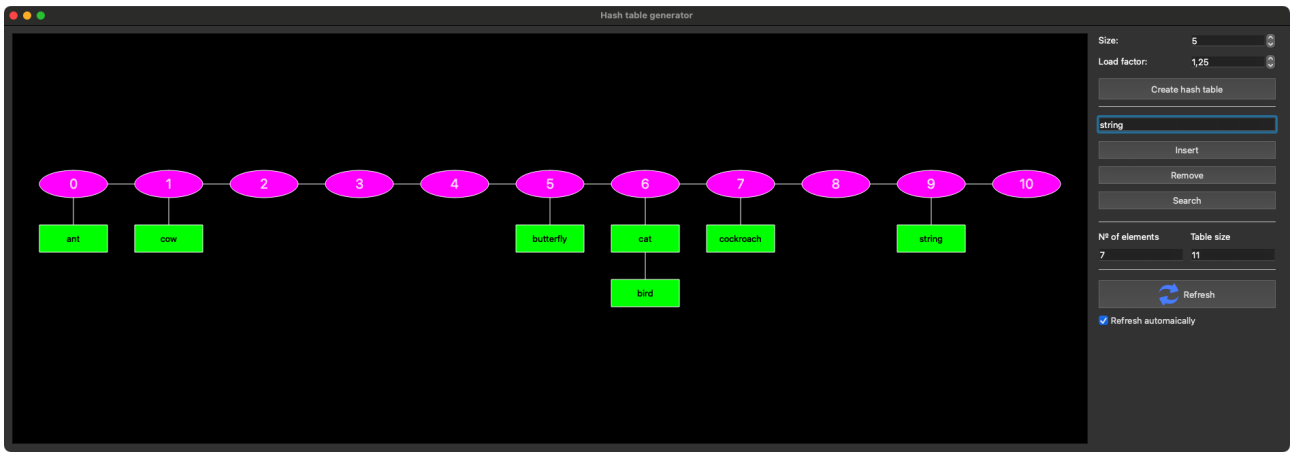


Рис. 2. Вставка елементу string

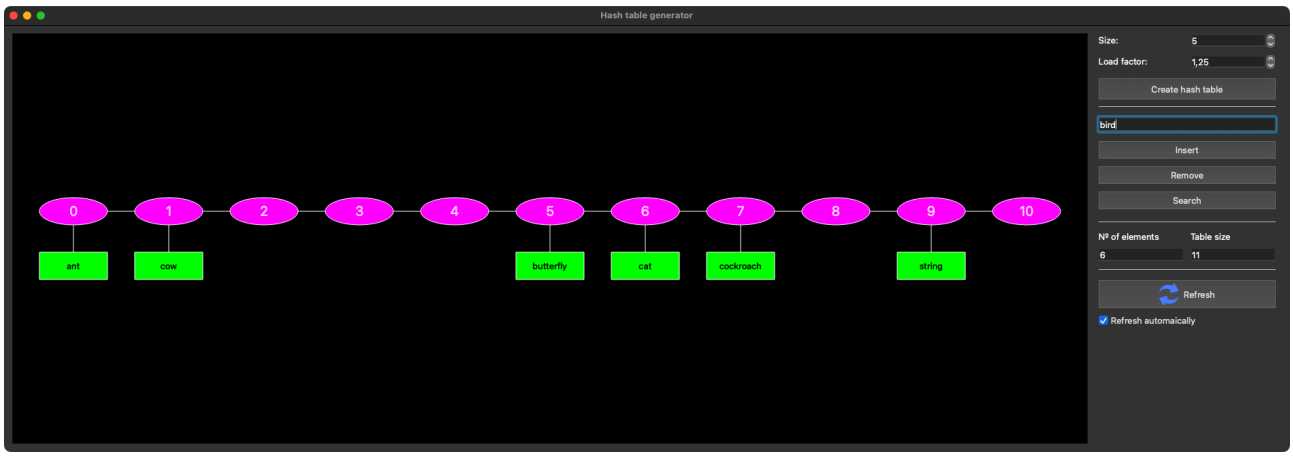


Рис. 3. Видалення елементу bird

# 6. Висновки

- Ознайомився:
  1. Хеш-таблицями.
  2. Хеш-функціями.
  3. Колізіями.
  4. Методами вирішення колізій.
- Вивчив:
  1. Основні алгоритми побудови хеш-функцій.
  2. Методи побудови хеш-таблиць і основні алгоритми пробірування.
- Реалізував:
  1. Графічний користувацький інтерфейс.
  2. Двохзв'язний список.
  3. Лнацюжкову хеш-таблицю.
  4. Обробку хеш-таблиці відповідно до запиту користувача.

# Перелік джерел посилань

[1] Narasimha Karumanchi. *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications, 2020.

[2] Ткачук В.М. *Алгоритми та структура даних*. Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016, с. 286.

[3] Панік Леонід Олександроович Ільман Валерій Михайлович Іванов Олександр Петрович. *АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ Навчальний посібник*. Дніпропетровський національний університет залізничного транспорту імені академіка В. Лазаряна, 2019, с. 134.