Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Алгоритми та складність

Лабораторна робота №1

“Реалізація ідеального хешування.”

Виконала студентка 2-го курсу

Групи ІПС-22

Челушкіна Валерія Олександрівна

Київ – 2025

**Завдання**

Реалізація ідеального хешування для рядків (порядок як у словнику).

**Теорія.**

Ідеальне хешування — це метод побудови хеш-таблиці, в якій немає колізій, тобто кожен ключ має унікальний індекс у таблиці. У найгіршому випадку відбувається О(1) звертань до пам’яті.

Існує 2 основні підходи до хешування: статичне, коли набір елементів фіксований і не змінюється, і динамічний, коли набір може змінюватися. Будемо розглядати статичний.

Один із найпоширеніших методів є двохрівневий підхід (двохфазне хешування).

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.

**Алгоритм.**

1. Отримати набір елементів, в даному випадку рядки.
2. Перевірити набір на повтори. Якщо вони є, усунути їх.
3. Створити хеш-таблицю. Розмір дорівнює кількості елементів у початковому наборі елементів.
4. Для кожного елементу з набору рахуємо за формулою:

h(i)=((a\*h(i-1)+b)mod p, i=0,n; h(0)=0; x=h(k)=h(n)mod m,

де a,b – певні значення, p – певне просте число, m – кількість елементів у наборі.

1. Якщо до однієї комірки потрапляє кілька елементів створюємо підхеш-таблицю у цій комірці, розмір якої дорівнює квадрату кількості елементів, що потрапили в одну й ту саму комірку.
2. Хешуємо кожний елемент підтаблиці попередньо обравши нову змінну x. Повторюємо доки не буде нових колізій.

**Складність.**

В загальному випадку алгоритм створення такої таблиці працює за час O(n). Для алгоритму пошуку у найгіршому випадку час дорівнюватиме O(1).

**Мова реалізації алгоритму.**

С++

**Модулі програми.**

* int primary\_hash(const string &key, int m) – функція для хешування першого рівня
* int secondary\_hash(const string &key, int m, int a2, int b2) – функція для хешування другого рівня
* PerfectHashTable(const vector<string> &keys) – конструктор для хеш-таблиці
* ~PerfectHashTable() – деструктор для хеш-таблиці
* SecondaryHashTable(const vector<string> &values) – конструктор для підхеш-таблиці
* int find(const string &key) – функція для знаходження за словом його місце в таблиці
* void print() – виведення таблиці на екран

**Інтерфейс користувача.**

Введення даних відбувається або користувачем, або береться із програми.

*Вхідні дані:*

вектор рядків

*Вихідні дані:*

хеш-таблиця

**Тестові приклади**

**Приклад**

Користувачем вводиться вектор рядків: ferret, parrot, hamster, rat, hedgehog, cat, dog

Розглянемо принцип хешування на прикладі слова rat. Обираємо значення a=2, b=47, p=211, m=7. Можемо використати формулу: h(i)=((a\*h(i-1)+char+b)mod p). h=h(i)\*mod m

Переводимо кожну літеру слова типу char до типу int і отримаємо: {114, 97, 116}.

За формулою хешуємо кожен символ окремо, покроково:

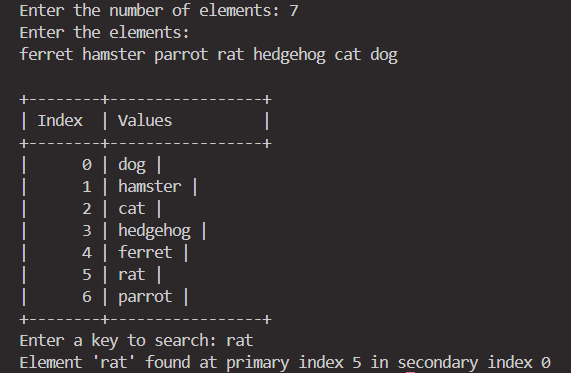
h(1)=(0\*2+114+47)mod 211=161

h(2)=(161\*2+97+47)mod 211=44

h(3)=(44\*2+116+47)mod 211=40

h=40mod7=5

Вийшов індекс 5, тобто наш елемент буде знаходитись у хеш-таблиці на 5 місці.



**Висновки.**

Ідеальне хешування використовується в: cистемах компіляції (збереження ключових слів мови програмування), oптимізованих структурах даних (наприклад, у файлових системах і базах даних), швидкому пошуку в статичних множинах (наприклад, словниках).

Ідеальне хешування є потужним методом для реалізації ефективних структур даних, що забезпечують постійний час доступу до елементів без колізій. Його використання доцільне у випадках, коли набір ключів є заздалегідь відомим або коли необхідно мінімізувати витрати часу на пошук.

**Література.**

* Лекція з предмету «Алгоритми та складність #1»
* [**https://en.wikipedia.org/wiki/Hash\_function**](https://en.wikipedia.org/wiki/Hash_function)
* [**https://en.wikipedia.org/wiki/Perfect\_hash\_function**](https://en.wikipedia.org/wiki/Perfect_hash_function)