Використовуємо алгоритм Флойда–Уоршелла для знаходження найсильніших шляхів між кандидатами у графі. Цей процес повторюється для всіх комбінацій кандидатів i, j та k, щоб гарантувати, що найсильніші шляхи враховують усі можливі проміжні кандидати. В результаті отримується матриця найсильніших шляхів між усіма кандидатами. Зрештою, програмно визначаємо переможця виборів за методом Шульце на основі матриці найсильніших шляхів між кандидатами.

Отже, метод Шульце є ефективним способом визначення переможця у виборах з великою кількістю кандидатів та виборців. Його реалізація за допомогою мови програмування JavaScript дає змогу легко використовувати цей метод у вебдодатках та інших проєктах, що вимагають проведення голосування.

Список використаних джерел

1. Schulze method. *Electowiki.* 14 July 2023. URL: https://electowiki.org/wiki/Schulze_method (дата звернення: 20.05.2024).

2. Schulze method. *Wikipedia.* 6 September 2024. URL: https://en.wikipedia.org/wiki/Schulze_method (дата звернення: 20.05.2024).

УДК 004.6

Поліщук О. С., *здобувачка 2 курсу спеціальності 122 Комп'ютерні науки,*
Поремський Ю. В., *канд. техн. наук, старший викладач кафедри інформаційних технологій*

ТЕОРЕТИЧНІ ЗАСАДИ ХЕШУВАННЯ ДАНИХ

Донецький національний університет імені Василя Стуса, м. Вінниця

Хешуванням даних називається операція перетворення вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини за допомогою деякого визначеного алгоритму. Перетворення вхідного масиву даних відбувається за допомогою спеціальної функції, яку називають хеш-функцією, або функцією згортки [1].

Вхідні дані, на які діє хеш-функція, називаються ключем (інколи повідомленням). Результат дії хеш-функції на вхідні дані називається хеш-значенням, хеш-кодом або просто хешем. Хеш – це натуральне число, яке часто записують у шістнадцятковому вигляді [1].

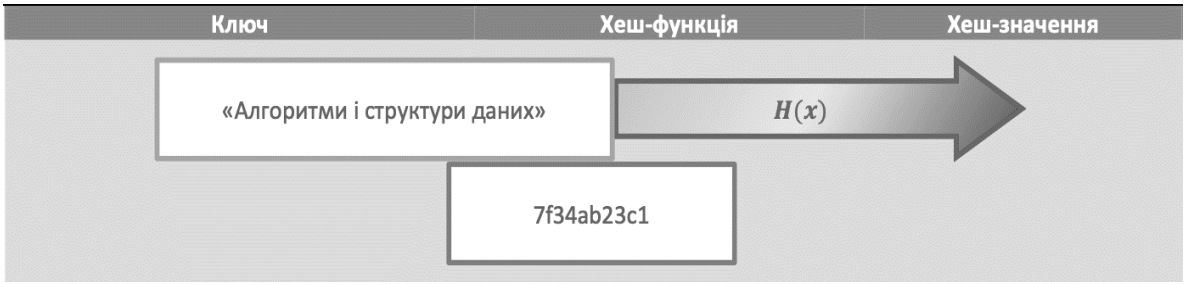

Рис. 1. Хешування рядка

Ідеальною хеш-функцією є така хеш-функція, яка для будь-яких двох неоднакових ключів дає неоднакові адреси. Підібрати таку функцію можна у випадку, якщо всі можливі значення ключів відомі наперед. Така організація даних має назву «досконале хешування».

Для реалізації хешування необхідні три речі:

- структура даних (хеш-таблиця) для зберігання даних;
- функція хешування, яка встановлює відповідність між значеннями ключа і розміщенням в таблиці;
- алгоритм вирішення конфліктів, який визначає послідовність дій, якщо декілька ключів відповідають одній комірці таблиці [2].

Простий приклад практичної реалізації хешування:

```
using System;
using System.Security.Cryptography;
using System.Text;

class Program
{
   static void Main(string[] args)
   {
      // Приклад хешування рядка "Hello, World!"
      string input = "Hello, World!";
      string hashedString = ComputeHash(input);
      Console.WriteLine($"Input: {input}");
      Console.WriteLine($"Hash: {hashedString}");
   }

   static string ComputeHash(string input)
   {
      // Створення екземпляру об’єкта MD5
      using (MD5 md5 = MD5.Create())
      {
         // Конвертування вхідного рядка у байтовий масив
         byte[] inputBytes = Encoding.UTF8.GetBytes(input);

         // Обчислення хеша для вхідних даних
         byte[] hashBytes = md5.ComputeHash(inputBytes);

         // Перетворення байтів хеша у рядок шістнадцяткового представлення
         StringBuilder sb = new StringBuilder();
         for (int i = 0; i < hashBytes.Length; i++)
         {
            sb.Append(hashBytes[i].ToString("x2"));
         }

         return sb.ToString();
      }
   }
}
```

```
Input: Hello, World!
Hash: 65a8e27d8879283831b664bd8b7f0ad4
```

Рис. 2. Результат програмного коду

Для розв’язання колізій використовуються різноманітні методи, які в основному зводяться до методів «ланцюжків» і «відкритої адресації».

Пошук у хеш-таблиці з ланцюжками переповнення здійснюється так. Спочатку обчислюється адреса за значенням ключа. Потім здійснюється послідовний пошук у списку, який зв’язаний з обчисленою адресою. Процедура вилучення з таблиці зводиться до пошуку елементу і його вилучення з ланцюжка переповнення.

Метод відкритої адресації полягає в тому, щоб, користуючись якимось алгоритмом, який забезпечує перебір елементів таблиці, переглядати їх у пошуках вільного місця для нового запису.

Очевидно, що в міру заповнення хеш-таблиці будуть відбуватися колізії, і внаслідок їх розв’язання методами відкритої адресації чергова адреса може вийти за межі адресного простору таблиці. Щоб це явище відбувалося рідше, можна піти на збільшення розмірів таблиці, порівняно з діапазоном адрес, які обчислюються хеш-функцією.

З одного боку, це приведе до скорочення кількості колізій і прискорення роботи з хеш-таблицею, а з іншого – до нераціональних витрат адресного простору. Навіть у разі збільшення таблиці удвічі, порівняно з областю значень хеш-функції, нема гарантій того, що внаслідок колізій адреса не перевищить розмір таблиці. Водночас у початковій частині таблиця може залишатися достатньо вільних елементів. Тому на практиці використовують циклічний перехід на початок таблиці.

Для того, щоб попередньо оцінити якість хеш-функції, можна провести імітаційне моделювання. Формується вектор цілих чисел, довжина якого співпадає з довжиною хеш-таблиці. Випадково генерується достатньо велика кількість ключів, для кожного ключа обчислюється хеш-функція. В елементах вектора підраховується кількість генерацій даної адреси. За результатами такого моделювання можна побудувати графік розподілу значень хеш-функції.

Якщо кількість елементів таблиці достатньо велика, то графік будується не для окремих адрес, а для груп адрес. Великі нерівномірності засвідчують високу імовірність колізій в окремих місцях таблиці. Зрозуміло, така оцінка є наближеною, але вона дає змогу попередньо оцінити якість хеш-функції і уникнути грубих помилок під час її побудови.

Оцінка буде більше точною, якщо генеровані ключі будуть більш близькими до реальних ключів, які використовуються під час заповнення хеш-таблиці. Для символьних ключів дуже важливо добитися відповідності генерованих кодів символів тим кодам символів, які є в реальному ключі. Для цього потрібно проаналізувати, які символи можуть бути використані в ключі.

Список використаних джерел

1. Креневич А. П. Алгоритми і структури даних: підручник. Київ: ВПЦ «Київський Університет», 2021. 200 с.

2. Воробйова О. Д., Глазунова Л. В. Алгоритми пошуку, стиснення даних, внутрішнього та зовнішнього сортування, алгоритми на графах. Одеса: ОНАЗ ім. О. С. Попова, 2017. 52 с.