Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Алгоритми та складність

Завдання №3

“ Узагальнення методу Рабіна-Карпа пошуку зразка в текстовому рядку”

Варіант №3

Виконав студент 2-го курсу

Групи ІПС-21

Тесленко Назар Олександрович

Київ – 2024

Завдання:

Узагальніть метод Рабіна-Карпа пошуку зразка в текстовому рядку так, щоб він дозволив розв’язати задачу пошуку заданого зразка розміром m на m у символьному масиві розміром n на n. Зразок можна рухати по горизонталі та вертикалі, але не обертати.

Теорія:

Алгоритм Рабіна-Карпа для пошуку зразка є ефективним методом, який базується на обчисленні та порівнянні хешів текстових рядків. Узагальнення цього алгоритму для двовимірного пошуку дозволяє використовувати його для знаходження шаблону m x m у символьному масиві n x n, де шаблон може переміщуватися горизонтально та вертикально без обертання. Він застосовується для швидкого пошуку шаблонів, і є популярним завдяки своїй ефективності, особливо для великих обсягів даних.

Шаблон (зразок) — це фрагмент, який ми намагаємося знайти. Двовимірний символьний масив розміром m × m, який порівнюється з різними підмасивами більшого символьного масиву (тексту).

Текст — це символьний масив, у якому здійснюється пошук. Це двовимірний масив розміром n × n, який представляє область, де шукаються всі можливі підмасиви розміром m × m.

Хешування — це метод перетворення даних (наприклад, рядка або двовимірної матриці) у фіксовану довжину числового значення (хеш), яке служить унікальним ідентифікатором для цих даних. Хеш-функції використовуються для прискорення пошуку, оскільки порівняння числових значень відбувається швидше, ніж порівняння всього масиву або рядка.

$$\boldsymbol{hash(S) = (S[0] \cdot p^{m-1} + S[1] \cdot p^{m-2} + \cdots + S[m-1] \cdot p^{0}) mod q}$$

де:

- S — це рядок, для якого обчислюється хеш;
- m — довжина рядка;

- p — деяке просте число (база хешування) ;
- q — модуль, щоб уникнути надмірно великих значень хешу.

База хешування- число, яке використовується як база для степеневих обчислень у хеш-функції. «p» визначає, як "важать" символи в рядках. Кожен символ в рядку зразка, залежно від позиції, має різну "силу", яка обчислюється як p у степені, що залежить від його місця в рядку.Його використання допомагає уникнути колізій.

Колізія хешів — це ситуація, коли дві різні підматриці або рядки мають однаковий хеш-код.

Ролінг-хеш (ковзне хешування) — це техніка, яка дозволяє оновлювати хеш для нового фрагмента даних на основі хешу попереднього фрагмента. Замість повного перерахунку хешу, під час зсуву вікна, старі елементи віднімаються з хешу, а нові додаються. Це дозволяє швидко обчислювати хеші для послідовних підмасивів або підрядків.

Зсув по рядках і стовпцях — це процес поступового переміщення вікна розміру m × m по символьному масиву з кроком на одну позицію вправо (по горизонталі) або вниз (по вертикалі), щоб перевірити відповідність зразку на кожній новій позиції.

Алгоритм:

Алгоритм пошуку зразка розміром m × m у символьному масиві розміром n × n за допомогою узагальненого методу Рабіна-Карпа складається з таких основних етапів:

Підготовка даних: спочатку визначається розмірність шаблону та тексту, обчислюються максимальні значення степенів для рядків і стовпців шаблону, які будуть використані для обчислення хешів

Обчислення початкових хешів:

- Для зразка і для початкової області тексту розміром m × m обчислюються хеші. Хеш для кожного стовпця обчислюється, послідовно перетворюючи кожен рядок зразка або тексту в числовий хеш-код.

Обчислення загального хешу зразка:

- Після обчислення хешів окремих стовпців, обчислюється загальний хеш для всього зразка шляхом зведення отриманих хешів стовпців у єдине значення. Це дозволяє представити зразок у вигляді одного числового хеш-коду.

Перевірка відповідності в поточній області тексту:

- Алгоритм починає з перевірки хешу початкової області тексту. Якщо хеш зразка збігається з хешем поточної області тексту, виконується детальне порівняння кожного символу зразка з відповідним символом у тексті для підтвердження збігу.

Ковзне хешування по стовпцях:

- Алгоритм зсуває вікно на один стовпець вправо. Для цього використовується техніка ковзного хешування, яка дозволяє ефективно оновлювати хеш, віднімаючи символи зліва, що вже були перевіреними, та додаючи нові символи з правого боку. Це дозволяє уникнути повного перерахунку хешу для кожної нової області.

Перевірка відповідності на нових стовпцях:

- Після оновлення хешу поточної області тексту виконується порівняння хеш-кодів. Якщо хеш збігається із хешем зразка, знову проводиться детальне порівняння символів.

Ковзне хешування по рядках:

- Після перевірки всіх стовпців у поточному рядку, вікно зсувається вниз на один рядок. Використовуючи ковзне хешування, хеші стовпців оновлюються таким чином, що верхні символи видаляються, а нові символи додаються знизу.

Повторення перевірки для всіх областей тексту:

- Описані дії повторюються для кожного нового рядка та стовпця символьного масиву до тих пір, поки не буде перевірена вся можливі області розміром m × m у тексті.

Результат:

- Якщо у тексті був знайдений результат, фіксуються координати верхньої лівої відповідної області тексту
- Після закінчення виконання алгоритму повертається вектор, що зберігає усі координати знайдених збігів

Даний алгоритм можна реалізувати таким псевдокодом:

1.Обчислення максимального степеню для рядків і стовпців шаблону:

maxColumnPower = обчислити (radix ^ (m - 1)) % mod

maxRowPower = обчислити (radix ^ (m - 1)) % mod

2.Обчислення хешів для шаблону та тексту

for кожного стовпця i від 0 до n:

rowHash = 0

for кожен рядок j від 0 до m:

rowHash = (rowHash * radix + символ тексту або шаблона)%mod

зберегти rowHash у список хешів

3.Обчислення початкового хешу для матриці шаблону:

patternMatrixHash = 0

for кожен стовпець від 0 до m:

patternMatrixHash = (patternMatrixHash * radix + хеш стовпця шаблона) % mod

4.Пошук шаблону в тексті

for кожен рядок i від 0 до (n - m):

textMatrixHash = 0

for кожен стовпець від 0 до m:

textMatrixHash = (textMatrixHash * radix + хеш стовпця тексту) % mod

for кожен стовпець j від 0 до (n - m):

if textMatrixHash == patternMatrixHash:

if перевірити символи шаблона і тексту:

додати координати (i, j) у список результатів

оновити хеш для наступного стовпця за допомогою ролінг хешу

5.Оновлення хешів для наступного рядка:

if i < (n - m):

for кожного стовпця:

оновити хеш рядка за допомогою ролінг хешу

повернути список результатів

Мова реалізації: C++

Модулі програми:

Головний клас:

- class RabinKarp

Реалізує алгоритм пошуку зразка в 2D матриці символів за допомогою хешування Рабіна-Карпа.

Методи класу RabinKarp:

- powerUnderMod(int number)

Обчислює степінь radix під модулем mod. Це потрібно для підтримки хеш-функції в алгоритмі, а саме для оновлення хешів

- findHash(vector<string>& matrix) const

Обчислює початкові хеші для рядків матриці (тексту або шаблону) і повертає їх у вигляді вектора хешів

- rollingHash(vector<long long>& textHash, long long& textMatrixHash, long row)

Оновлює хеш тексту при зсуві шаблону по горизонталі, використовуючи метод "ковзаючого" хешування

- collumnRollingHash(vector<string>& text, vector<long long>& textHash, int row)

Оновлює хеші для стовпців тексту при переміщенні шаблону по вертикалі.

- check(vector<string>& text, vector<string>& pattern, int row, int column)

Перевіряє, чи відповідає шаблон тексту в конкретній позиції (починаючи з координат [row, column] що ітеруються в for-циклах)

- rabinKarpSearch(vector<string>& text, vector<string>& pattern)

Головний метод, що виконує пошук шаблону в тексті, використовуючи хешування за методом Рабіна-Карпа і повертає індекси, де знайдено шаблон.

Складність алгоритму:

Метод **powerUnderMod(int number)**

Часова складність:

Під час кожної рекурсії, число ділиться на два. Таким чином, кількість рекурсій — це логарифм від числа: $\boldsymbol{O(\log{(n)})}$

- **Загальна складність:** $O(\log(n))$

Метод **findHash(vector<string>& matrix)**

Часова складність:

Зовнішній цикл по стовпцях: O(n)

Внутрішній цикл по рядках (кількість рядків шаблона = patternRows):O(m)

Обчислення хешу для кожного символа: O(1)

- **Загальна складність**: $O(m * n)$

Метод **rollingHash(vector<long long>& textHash, long long& textMatrixHash, long row)**

Часова складність:

Операція віднімання хешу рядкаO(1).

Операція множення для пересування хешу: O(1).

Операція додавання нового хешу рядка (якщо є): O(1)

- **Загальна складність**: $O(1)$ – оскільки просто проводимо певні математичні операції над вже відомими даними

Метод **collumnRollingHash(vector<string>& text, vector<long long>& textHash, int row)**

Часова складність:

Зовнішній цикл по стовпцях: O(n)

Операція віднімання старого хешу: O(1)

Операція множення для пересування хешу: O(1)

Операція додавання нового хешу рядка: O(1)

- Загальна складність: $O(n)$

Метод **check(vector<string>& text, vector<string>& pattern, int row, int column)**

Часова складність:

Зовнішній цикл по рядках шаблона: O(m)
Внутрішній цикл по стовпцях шаблона: O(m)
Перевірка відповідності кожного елемента: O(1)

- **Загальна складність: $\boldsymbol{O(n^2)}$**

Метод **rabinKarpSearch(vector<string>& text, vector<string>& pattern)**

Часова складність:

1. **Обчислення початкових хешів для тексту та шаблону:**
   - Виклик методу findHash для тексту: **O(mn)**, де n — кількість стовпців у тексті, m — кількість рядків у шаблоні
   - Виклик методу findHash для шаблону: $\boldsymbol{O(m^2)}$
2. **Обчислення початкового хешу матриці шаблона:**
   - Зовнішній цикл по стовпцях шаблона: O(m)
   - Обчислення загального хешу для шаблона: O(1) для кожного стовпця

   Складність: O(m)
3. **Основний цикл пошуку:**
   - Зовнішній цикл по рядках тексту (від 0 до n−m): O(n−m), де n — кількість рядків тексту
   - Внутрішній цикл по стовпцях тексту (від 0 n−m): O(n−m), де n — кількість стовпців
   - Порівняння хешів для шаблона і поточної підматриці тексту: O(1).
   - Якщо хеші співпадають, викликається метод check, який має складність $O(m^2)$ (перевірка кожного символа шаблона)

   Складність: $\boldsymbol{O(1) + O(m^2)}$ у випадку збігу хешів
4. **Оновлення хешів (rolling hash):**
   - Для кожного рядка тексту обчислюється новий хеш: O(1)
   - Для кожного стовпця тексту обчислюється новий хеш за допомогою методу collumnRollingHash: $O(n)$

Складність алгоритму для одного рядка тексту:

- У гіршому випадку, для кожної підматриці тексту потрібно виконати порівняння символів за $\boldsymbol{O(m^2)}$

Загальна часова складність:

У гіршому випадку (коли необхідна перевірка символів для кожної підматриці) за умови, що ми маємо квадратні матриці n x n, m x m, як для тексту так і для шаблону отримаємо:

Часова складність: $O\big((n-m)(n-m)*m^2\big) => \boldsymbol{O((n-m)^2 * m^2)}$

Але алгоритм коректно працюватиме для випадків коли шаблон та текст будуть задані не квадратними матрицями отже розрахуємо часову складність й для цього випадку:

Нехай:

N1 = textRows,

N2 = textColumns,

M1 = patternRows,

M2 = patternColumns

Часова складність: $\boldsymbol{O\big((N1-M1)*(N2-M2)*(M1*M2)\big)}$

Інтерфейс користувача:

Введення даних користувачем відбувається через консоль. Спочатку вводиться розмір текстової матриці:

- Кількість рядків текстової матриці: n
- Кількість стовпчиків текстової матриці: n

Далі вводимо по рядкам саму текстову матрицю

Після цього вводяться розміри матриці шаблону:

- Кількість рядків матриці шаблону: m
- Кількість стовпчиків матриці шаблону: m

Далі вводимо по рядкам сам шаблон:

Після чого отримуємо список верхніх лівих точок зафіксованих у тексті

Тестові приклади:

Тест 1

Для першого прикладу візьмемо текст розміром 4*4:

ABCD

EFGH

IJKL

MNOP

і шаблон розміром 2*2:

FG

JK

```
Enter the number of rows for the text: 4
Enter the number of columns for the text: 4
Enter the text matrix row by row:
ABCD
EFGH
IJKL
MNOP
Enter the number of rows for the pattern: 2
Enter the number of columns for the pattern: 2
Enter the pattern matrix row by row:
FG
JK
Pattern found at the following coordinates (row, column):
(1, 1)
```

Як можемо бачити алгоритм знайшов шаблон у тексті за координатами (1;1)
Виконання алгоритму успішне

Тест 2

Візьмемо текст розміром 3*3, а шаблон розміром 2*2:

```
Enter the number of rows for the text: 3
Enter the number of columns for the text: 3
Enter the text matrix row by row:
QQI
JQQ
NJQ
Enter the number of rows for the pattern: 2
Enter the number of columns for the pattern: 2
Enter the pattern matrix row by row:
QQ
JQ
Pattern found at the following coordinates (row, column):
(0, 0)
(1, 1)
```

Як можемо бачити, у тексті були знайдені 2 приклади шаблону, що вказані у реузльтаті в координатах

Тест 3

Візьмемо текст розміром 8*8:

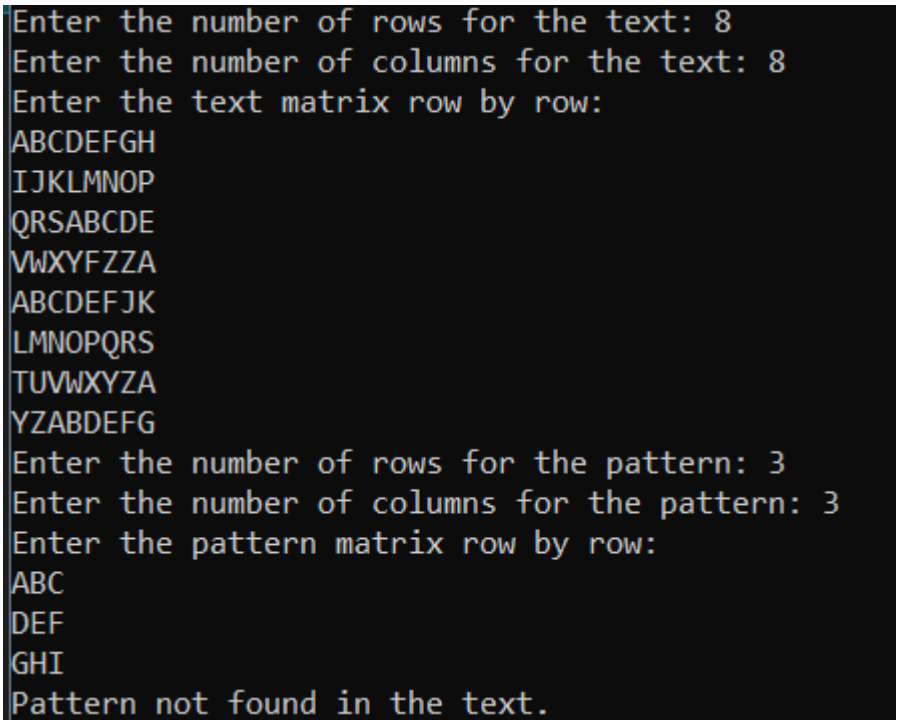
ABCDEFGH

IJKLMNOP

QRSABCDE

VWXYFZZA

ABCDEFJK
LMNOPQRS
TUVWXYZA
YZABDEFG
І шаблон розміру 3*3:
ABC
DEF
GHI



Алгоритм спрацював вірно адже подібних шаблонів у тексті немає

Тест 4

Також можемо взяти не квадратні мтариці великих розмірів, як для тексту, так і для шаблону. Отже розмір матриці тексту буде 50*42, а шаблону 4*6.

Шаблон:

AAAAAA
BBBBBB
CCCCCC
DDDDDD

```
Enter the number of rows for the text: 50
Enter the number of columns for the text: 42
Enter the text matrix row by row:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXAAAAAAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXBBBBBBXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXCCCCCCXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXDDDDDDXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXAAAAAAXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXBBBBBBXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXCCCCCCXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXDDDDDDXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXAAAAAAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXBBBBBBXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXCCCCCCXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXDDDDDDXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXAAAAAAXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXBBBBBBXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXCCCCCCXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXDDDDDDXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
Enter the number of rows for the pattern: 4
Enter the number of columns for the pattern: 6
Enter the pattern matrix row by row:
AAAAAA
BBBBBB
CCCCCC
DDDDDD
Pattern found at the following coordinates (row, column):
(1, 6)
(6, 24)
(16, 4)
(28, 23)
```

Можемо бачити, що шаблон у тексті зустрічається 4 рази, що підтверджує коректність результатів наданих алгоритмом

Отже виконавши алгоритм для різних умов, а саме використовуючи різний розмір матриць та не квадратні матриці, можемо переконатись що метод Рабіна-Карпа для пошуку зразка у тексті працює коректно

Висновки:

У даній роботі було розглянуто алгоритм Рабіна-Карпа для пошуку шаблону у двовимірному масиві. Алгоритм був узагальнений для роботи з матрицями, що дозволяє знаходити заданий шаблон розміру m x m у тексті розміром n x n, використовуючи хешування для прискорення пошуку. У процесі було реалізовано програму, що виконує пошук шаблону, оптимізуючи його за допомогою ролінг хешування, яке знижує кількість необхідних порівнянь та робить пошук більш ефективним.

Під час розробки алгоритму було враховано обчислювальну складність операцій, що дозволило знизити загальну складність алгоритму до $\boldsymbol{O}((\boldsymbol{n}-\boldsymbol{m})^{\boldsymbol{2}} * \boldsymbol{m}^{\boldsymbol{2}})$, де n — розмір тексту, а m — розмір шаблону. Це робить алгоритм ефективним для великих матриць та дозволяє знайти всі можливі входження шаблону в текст за прийнятний час.

Використані джерела:

- Алгоритм Рабіна-Карпа
- Pattern Searching with Rabin-Karp Algorithm
- Відео: rabin-karp in 60 seconds
- Rabin-Karp algorithm