Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Алгоритми та складність

Завдання № 3
«Двовимірний пошук підрядка (2D-алгоритм Рабіна–Карпа)»

Виконав: студент 2-го курсу
групи ІСП-21
Ювженко Назарій Олександрович

Київ – 2025

Завдання

Реалізувати алгоритм двовимірного пошуку зразка (pattern) у тексті, що подано у вигляді прямокутної таблиці символів (рядків однакової довжини), з використанням 2D-варіанту алгоритму Рабіна–Карпа.

Алгоритм має:

1. Побудувати хеш-таблицю для всього тексту (двовимірний префікс-хеш).
2. Побудувати хеш для зразка.
3. Перебрати всі можливі позиції зсуву зразка всередині тексту.
4. На кожній позиції швидко (O(1)) одержувати хеш відповідного підквадрата тексту та порівнювати з хешем зразка.
5. У випадку збігу хешів виконати покрокову перевірку символів, щоб виключити зіткнення (колізії) хешів.
6. Вивести всі координати входжень зразка у текст.

В якості тестових даних використати символну матрицю розміру 5×5 і зразок 2×2 з наведеного коду.

Теорія

Алгоритм Рабіна–Карпа у класичному варіанті використовує ідею порівняння хешів, а не самих рядків: якщо хеш підрядка тексту дорівнює хешу зразка, то з великою ймовірністю підрядки збігаються, і тоді виконується остаточна перевірка символів.

Для 2D-випадку ми працюємо вже не з рядком, а з прямокутником (квадратом) символів. Ідея та сама:

- спочатку обчислюємо двовимірний префіксний хеш для всього тексту;
- далі вміємо за O(1) дістати хеш будь-якого підпрямокутника;
- обчислюємо хеш зразка тим самим способом;
- порівнюємо ці значення.

Щоб хеш був однозначним для двох вимірів, у коді використовуються дві різні основи (bases):

- BASE_ROW — для переходу між рядками;

- BASE_COL — для переходу між стовпцями;
  а також велике просте число MOD = 1 000 000 007 для приведення значень по модулю.

Формула для побудови 2D-префіксного хеша (функція buildHash2D) реалізує аналог двовимірної формули включень–виключень:

H[i][j] = val

+ H[i-1][j] * BASE_ROW

+ H[i][j-1] * BASE_COL

- H[i-1][j-1] * BASE_ROW * BASE_COL

де

- val — числове значення символу в клітинці (береться як unsigned char),
- перші два доданки “притягують” зверху та зліва вже пораховані підхеші,
- останній доданок віднімаємо, бо він був доданий двічі (зверху і зліва),
- усе береться по модулю MOD.

Щоб з підхеша “вирізати” прямокутник однакового розміру (m×m), нам також потрібні таблиці степенів для обох основ — вони будуються у функції buildPows. Саме тому getSubHash(...) отримує і хеш-таблицю, і обидві таблиці степенів.

3. Опис алгоритму

3.1. Побудова таблиць степенів

vector<long long> powRow = buildPows(BASE_ROW, n);

vector<long long> powCol = buildPows(BASE_COL, n);

- buildPows(base, n) повертає вектор p[0..n], де p[i] = base^i mod MOD.
- Ці таблиці потрібні, щоб при “вирізанні” підпрямокутника з великого хеша ми могли правильно домножити й відняти частини.

3.2. Побудова 2D-хеша для тексту і для зразка

vector<vector<long long>> Htext = buildHash2D(text);

vector<vector<long long>> Hpat = buildHash2D(pattern);

- Для тексту розміру n×n отримаємо таблицю Htext[n][n].

- Для зразка розміру m×m — таблицю Hpat[m][m].

3.3. Отримання хеша зразка

```
long long patHash = getSubHash(Hpat, powRow, powCol, 0, 0, m);
```

- Оскільки зразок повністю збігається з власною хеш-таблицею, ми просто беремо підпрямокутник із кутом (0,0) розміру m.

3.4. Перебір усіх позицій у тексті

```
for (int i = 0; i + m <= n; i++)
   for (int j = 0; j + m <= n; j++) {
      long long cur = getSubHash(Htext, powRow, powCol, i, j, m);
      ...
   }
```

- Ми зміщуємо вікно розміру m×m по всьому тексту.
- Для кожного вікна беремо хеш за O(1).

3.5. Порівняння і верифікація

```
if (cur == patHash) {
   // покрокова перевірка символів
}
```

- Якщо хеші рівні, це *ймовірно* збіг.
- Щоб виключити колізії, виконується подвійний цикл:
- for (int x = 0; x < m && ok; x++)
- for (int y = 0; y < m; y++)
- if (text[i+x][j+y] != pattern[x][y]) ok = false;
- Лише якщо всі символи збіглися — фіксуємо входження.

3.6. Вивід результату

```
cout << "found at (" << r << ", " << c << ")\n";
```

- Координати — верхній лівий кут входження.
- У нашому прикладі це будуть позиції, де у тексті підряд ідуть GH і під ними LM.

Таким чином, повний алгоритм — це “2D-префіксні хеші + вибірка підпрямокутника + верифікація”.

4. Опис реалізованих функцій

Нижче — відповідність між кодом і логічними модулями звіту.

1. **buildPows(long long base, int n)**
    - Призначення: попередньо обчислити степені основи base до n.
    - Використовується для нормалізації підхешів у getSubHash.
2. **buildHash2D(const vector<string>& a)**
    - Призначення: побудова двовимірної хеш-таблиці для рядків однакової довжини.
    - Реалізує формулу включень–виключень у 2D.
    - Повертає H[i][j] — хеш підпрямокутника з (0,0) до (i,j).
3. **getSubHash(...)**
    - Вхід: 2D-хеш H, таблиці степенів, верхній лівий кут (x,y) і розмір sz.
    - Вихід: нормалізований хеш підквадрата розміру sz×sz.
    - Виконує віднімання “зайвих” частин і домноження на степені, аналогічно до 2D-префіксних сум.
4. **rabinKarp2D(const vector<string>& text, const vector<string>& pattern)**
    - Головна алгоритмічна функція.
    - Створює хеші для тексту і зразка.
    - Перебирає всі позиції у тексті.
    - Порівнює хеші і, якщо треба, перевіряє символи.
    - Повертає список пар (i, j) — усі знайдені входження.
5. **main()**
    - Задає вхідні дані прямо в коді (без введення з клавіатури):
    - vector<string> text = {
    - "abcde",

- "fGHij",
- "kLMno",
- "psGHt",
- "uxLMy" };
- vector<string> pat = {
- "GH",
- "LM" };
- Викликає алгоритм і виводить координати.

5. Інтерфейс користувача

У поточному варіанті програма консольна і не вимагає введення від користувача: усі дані задані в коді у вигляді векторів рядків.

Вивід — у стандартний потік cout, формат:

found at (рядок, стовпець)

За потреби інтерфейс легко розширити:

- зчитувати розміри n, m;
- зчитувати n рядків тексту;
- зчитувати m рядків зразка;
- виводити кількість входжень і самі координати.

6. Тестовий приклад

Вхідні дані в коді:

Текст (5×5):

abcde

fGHij

kLMno

psGHt

uxLMy

Зразок (2×2):

GH

LM

Аналіз:

- У тексті на позиції (1,1) (рахуємо з 0) стоїть літера G, далі H, а під ними — L та M.
- Далі внизу, починаючи з (3,2), також зустрічається така ж комбінація:
  - рядок 3, стовпець 2 → G
  - рядок 3, стовпець 3 → H
  - рядок 4, стовпець 2 → L
  - рядок 4, стовпець 3 → M

Тому програма повинна вивести дві позиції входження:

found at (1, 1)

found at (3, 2)

(Порядок виводу залежить від порядку перебору, у нашому коді — спочатку по рядках, потім по стовпцях.)

7. Оцінка складності

Позначимо:

- N — розмір сторони тексту (для простоти беремо квадратний випадок N×N);
- M — розмір сторони зразка (M×M), $M \le N$.

1. **Побудова 2D-хеша тексту** — $O(N^2)$.
2. **Побудова 2D-хеша зразка** — $O(M^2)$ (незначно порівняно з N^2).
3. **Перебір усіх позицій** — можливих позицій $(N - M + 1)^2 \approx O(N^2)$.
4. **Отримання хеша підквадрата** через getSubHash(...) — O(1).
5. **Додаткова перевірка символів** — виконується тільки коли хеш збігся. У середньому це рідко, тому амортизовано алгоритм працює за $O(N^2)$.

6. У гіршому випадку, коли всі хеші збігаються (спеціально підібраний тест), отримаємо $O(N^2 \cdot M^2)$, але для нормальних даних це практично не трапляється.

Отже, очікувана складність:
$O(N^2)$ — на побудову й пошук.

Це й є головна перевага 2D-Рабіна–Карпа над “наївним” 2D-порівнянням, яке було б $O(N^2 \cdot M^2)$.

8. Висновки

У роботі реалізовано двовимірний варіант алгоритму Рабіна–Карпа для пошуку зразка в тексті, поданому у вигляді матриці символів. Було:

- побудовано дві таблиці степенів для різних основ (по рядках і по стовпцях);
- реалізовано побудову двовимірного префіксного хеша;
- реалізовано функцію вирізання хеша довільного підквадрата за O(1);
- здійснено порівняння хешу зразка з хешами всіх підквадратів тексту;
- додано фінальну перевірку символів для усунення колізій;
- протестовано роботу на прикладі з рядками "abcde" ... "uxLMy" і зразком ["GH","LM"].

Отриманий алгоритм показує очікуваний результат і має квадратичну від довжини сторони тексту складність, що відповідає вимогам до ефективних 2D-пошуків. Структура звіту та підхід до опису взяті за аналогією з наданим прикладом звіту з теми про алгоритм Джонсона.

Використані літературні джерела

- «Алгоритми та складність». Лекція №10
- Cormen T., Leiserson C., Rivest R., Stein C. *Introduction to Algorithms*
- Матеріали з хешування та префіксних сум (двовимірних)
- https://uk.wikipedia.org/wiki/Алгоритм_Рабіна_—_Карпа