e m

Rolling Hash

TAI, WEI-HSUAN

August 1, 2025

Contents

Abstract

Abstract

本講義旨在深入探討 Rolling Hash 字串搜尋演算法。Rolling Hash 演算法是一種基於 Hash 技術的字串搜尋方法，能夠在大規模文本中快速定位模式串的位置。

1 引言

字串搜尋是電腦科學中一個非常基礎且重要的問題：如何在一個長字串（文本，T）中找到一個短字串（模式串，P）的所有出現位置？最直觀的方法是樸素（Naive）演算法，但其在某些情況下效率較低。KMP 演算法正是為了解決這個效率問題而提出。

1.1 樸素字串搜尋演算法的缺陷

考慮在文本 $T =$ `ABABCABAB` 中搜尋模式串 $P =$ `ABCABAB`。

- 樸素演算法從 T 的開頭開始，逐一比較 P 和 T 的子字串。
- 如果在某個位置不匹配，則將 P 向右移動一位，重新從 P 的開頭與 T 的當前位置進行比較。

例如：

- T: `A B A B C A B A B`
- P: `A B C A B A B`
- 第一次匹配：`T[0]` 與 `P[0]` 匹配 (`A=A`)，`T[1]` 與 `P[1]` 匹配 (`B=B`)。
- `T[2]` (`A`) 與 `P[2]` (`C`) 不匹配。此時，樸素演算法會將 P 向右移動一位，從 `T[1]` 重新開始比較。

這種「回溯」行為會導致大量的重複比較，尤其是在模式串中存在重複字元時，效率會非常低。其最差時間複雜度可達 $O(m \times n)$，其中 n 是文本長度，m 是模式串長度。

2 Rolling Hash 演算法

2.1 概述

在前綴和的題目中，我們可以透過預處理來快速計算子字串的和。類似地，Rolling Hash 演算法利用雜湊函數來快速計算子字串的雜湊值，從而實現高效的字串匹配。

Rolling Hash 的核心思想為，使用雜湊 (Hash) 的方式將一個字串紀錄為一個整數，透過比較整數來判斷字串是否相等。這樣，我們可以在 $O(1)$ 的時間內比較兩個字串是否相等。由於 Hash 函數的特性，我們可以保證不同的字母在不同位置的組合會產生不同的雜湊值。如此一來，可以最大程度的保證不同字串雜湊值的唯一性。

2.2 先備知識

2.2.1 費馬小定理

已知 p 是一個質數，a 是一個正整數且不是 p 的倍數，則有：

$$a^p \equiv a \mod p$$

這意味著 a^p 除以 p 的餘數等於 a。

2.2.2 模逆元

已知 p 是一個質數，a 是一個正整數且不是 p 的倍數，則存在一個整數 $b = a^{p-2} \mod p$，使得：

$$a \cdot b \equiv 1 \mod p$$

這個整數 b 稱為 a 在模 p 下的逆元，通常用符號 a^{-1} 表示。

2.3 Hash 函數

在 Rolling Hash 中，我們使用一個多項式雜湊函數來計算字串的雜湊值。對於字串 $S = s_0 s_1 \dots s_{m-1}$，其雜湊值定義為：

$$H(S) = \sum_{i=0}^{|S|-1} s_i \cdot x^i \mod p$$

其中 S 是字串的長度，s_i 是字串中第 i 個字母轉換的編號，x 是一個正整數，p 是一個質數。使用這種方式計算 Hash ，我們可以確保每個字母在不同位置的組合會產生不同的雜湊值，進而確保不同字串雜湊值的唯一性。

讀者看到這邊，可能會感到疑惑：為什麼透過計算字串的雜湊值可以快速查找子字串？

透過 Hash 的方式，我們可以把一個字串映射到唯一的一個整數，因此，若我們能快速查找某個字串當中子字串的雜湊值，是否就能快速查找子字串是否存在於某個字串當中呢？

2.4 Rolling Hash 前綴和

為了能在 $O(1)$ 的時間內計算任意子字串的雜湊值，我們可以使用前綴和的思想。具體來說，我們可以預先計算出文本 T 的每個前綴的湊值，然後利用這些前綴雜湊值來快速計算任意子字串的雜湊值。

假設我們有字串 T，$T[a, b]$ 代表字串 T 中從位置 a 到位置 b 的子字串，為了能在 $O(1)$ 的時間內找出 $H(T[a, b])$，我們可以先進行一個 $O(T)$ 的預處理，定義 $S[N] = H(T[0, N])$，則透過以下的步驟：

$$S[0] = T[0]$$

$$S[i] = (S[i-1] + T[i] \cdot x^i) \mod p$$

可以在 $O(T)$ 的時間內計算出 S 完成預處理。當我們要查找 $H(T[a,b])$ 時，只需要透過以下的公式：

$$S[b]-S[a-1] \mod p = \sum_{i=0}^{b-1} T_i x^i - \sum_{i=0}^{a-1} T_i x^i \mod p = \sum_{i=a}^{b-1} T_i x^i \mod p = x^a \cdot H(T[a,b])$$

即可在 $O(1)$ 的時間內計算出 $H(T[a,b])$。
方才的公式得到的 $H(T[a,b])$ 並不乾淨，還多了一個 x^a 的因子，因此我們需要計算 x^a 的模逆元 $x^{-a} \mod p = (x^a)^{p-2} \mod p$，然後將其乘上 $S[b]-S[a-1]$，即可得到乾淨的 $H(T[a,b])$。
最後，可以得到：

$$H(T[a,b]) = x^{-a} \cdot (S[b]-S[a-1]) \mod p$$

2.5 Rolling Hash 碰撞分析

雖然 Rolling Hash 可以在 $O(1)$ 的時間內計算任意子字串的雜湊值，但仍然存在碰撞的可能性，即不同的字串可能會有相同的雜湊值。這是由於 Hash 函數的特性所導致的。為了減少碰撞的機率，我們可以選擇一個較大的質數 p 和一個合適的基數 x。
碰撞發生的概率不大，但仍然存在。為了進一步降低碰撞的概率，我們可以使用雙重雜湊（Double Hashing）技術，即對同一個字串使用兩個不同的雜湊函數，只有當兩個雜湊值都相等時，才認為兩個字串相等。

2.6 C++ 實作

給定一個字串 T 和一個模式串 P，求出 P 在 T 中出現的次數。

Listing 1: Rolling Hash 演算法的 C++ 實作

```cpp
vector<int> prepross(string s){
    vector<int> pre(s.size()+5);

    pre[0]=(int)(s[0]-'a');
    int now_x=1;
    for(int i=1;i<s.size();++i){
        now_x=now_x*x%p;
        pre[i]=(pre[i-1]+(int)(s[i]-'a')*now_x)%p;
    }
    return pre;
}

int query(vector<int> pre, int a, int b){
    if(a==0)return pre[b];
    int inverse=fpow(fpow(x, a), p-2);
    return ((pre[b]-pre[a-1]+p)*inverse)%p;
```