**Thuật toán Rabin – Karp**

**I – Giới thiệu thuật toán Karp – Rabin**

Thuật toán mang tên hai nhà khoa học phát minh ra nó Michael O.Rabin (sinh năm 1931, người Đức) và Richard M.Karp (sinh năm 1931, người Mỹ), đều được giải Turning Award, giải thưởng uy tín nhất trong ngành khoa học máy tính và công nghệ thông tin.

**II – Mô tả thuật toán**

Thuật toán Rabin Karp là một thuật toán được sử dụng để tìm kiếm chuỗi con pattern trong chuỗi cha text bằng cách sử dụng một hàm băm.

Hàm băm là một hàm chuyển đổi chuỗi thành giá trị số, giá trị này được gọi là mã băm của nó. Ví dụ, chúng ta có thể có hàm băm hash(“hello”) = 5

Giống như thuật toán Naive, thuật toán Rabin Karp cũng dịch pattern qua từng phần tử trong text để so sánh. Nhưng sự khác biệt là thuật toán Rabin Karp so khớp mã băm của pattern với mã băm của chuỗi con subtext của text, và nếu mã băm khớp thì thuật toán sẽ so sánh từng ký tự trong hai chuỗi với nhau.

Nếu mã băm được biểu diễn là số nguyên không quá 64 bits, độ phức tạp thời gian (time – complexity) của việc so sánh pattern có độ dài m với subtext có cùng độ dài giảm từ O(m) xuống O(1).

Tuy nhiên mọi thứ đều có hai mặt, vấn đề của hàm băm đó là mã băm của hai chuỗi khác nhau có thể bằng nhau. Ví dụ xét hàm băm hash(S), tính mã băm của xâu S bằng cách cộng mã ASCII của các kí tự trong S: hash(“abcd”) = 97 + 98 + 99 + 100 = 394, hàm băm này quá đơn giản và có khả năng gây trùng mã băm cao: hash(“dacb”) = 100 + 97 + 99 + 98 =394, nhưng “abcd” ≠ “dacb”.

Một hàm băm tốt thỏa mãn các điều kiện sau:

* Tính toán nhanh.
* Xác suất trùng mã băm nhỏ.

Thuật toán Rabin Karp xây dựng hàm băm với ý tưởng cơ số: xem mọi xâu như một chuỗi số với một cơ số (base) nào đó. Hàm băm được tính tương tự với việc ta chuyển một số nguyên về giá trị của nó, nếu là xâu kí tự thì có thể sử dụng mã ASCII (hoặc UNICODE). Một số ví dụ:

base 10, hash(“425”) = 4 × 102 + 2 × 101 + 5 × 100 = 425.

base 26, kí tự là chữ cái từ a đến z : hash (“abc”) = 97 × 262 + 98 × 261 + 99 × 260 = 68219

Để tránh tràn số thì kết quả trên được chia lấy dư cho một số q, thường được chọn là một số nguyên tố lớn. Nếu gọi tập các kí tự được sử dụng trong chuỗi là Q thì base thường được chọn sao cho base = hoặc là một số nguyên tố lớn.

Độ phức tạp thời gian để tính mã băm của chuỗi độ dài k mất O(k). Khi thực hiện thuật toán, ta sẽ “trượt ” pattern có độ dài m trên text từ vị trí 0 đến n-m để so sánh mã băm. Rolling Hash là hàm băm có thể tính mã băm hi của text[i…i+m-1] dựa trên mã băm hi – 1 của text[(i-1)…(i+m)] chỉ trong thời gian O(1) thay vì tính lại trong thời gian O(m), từ đó tăng tính hiệu quả

hi = (base × (hi -1 – basem-1 × text[i – 1]) + text[i + m – 1]) %q

**III – Cách cài đặt thuật toán**

**Text, letter

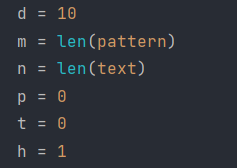
Description automatically generated**Mã giả cho giải thuật Rabin – Karp:

Đầu tiên khai báo hàm :



Với pattern là chuỗi cần tìm, text là chuỗi cha, q là số nguyên tố lớn, d là base

Ta khai báo các biến cần sử dụng:



Với d là cơ số, m là độ dài của chuỗi cần tìm và n là độ dài chuỗi cha. Còn p và t là 2 biến để lưu giá trị hash của chuỗi.

Graphical user interface, text

Description automatically generatedTính giá trị của base m-1:

Tính giá trị hash đầu tiên của chuỗi cha và hash của chuỗi cần tìm

A screenshot of a computer

Description automatically generated with medium confidence

Sau đó nếu ta tính được giá trị hash nào đó khi xét trong chuỗi cha bằng chuỗi con thì ta sẽ xét từng kí tự một, nếu đúng thì xuất ra vị trị của chuỗi cần tìm trong chuỗi cha. Còn nếu không thì ta tiếp tục tính giá trị hash tiếp theo trên chuỗi cha.

A screenshot of a computer

Description automatically generated with medium confidence

**IV – Phân tích độ phức tạp**

Với chuỗi pattern có độ dài M và chuỗi text có độ dài M:

Phân tích độ phức tạp thuật toán:

Trong quá trình tiền xử lý, để tính mã băm cho pattern cần duyệt qua m phần tử, tương tự với subtext đầu tiên text, vậy độ phức tạp là *O*(M+M) ∈ *O*(M).

Trong trường hợp tốt nhất, ta có thấy pattern chính là subtext đầu tiên của text, khi đó chỉ cần tốn M lần so sánh các ký tự, kết hợp với quá trình tiền xử lý nên độ phức tạp là *O*(M+M) ∈ *O*(M).

Trường hợp xấu nhất:

* Mỗi lần so sánh với subtext, pattern phải so sánh nhiều nhất là M lần (trong trường hợp mã băm bằng nhau và cả M – 1 ký tự đầu đều đúng).
* Có tất cả N – M – 1 chuỗi subtext, vậy số chuỗi cần so sánh nhiều nhất là N – M + 1 subtext như vậy (trong trường hợp chuỗi cần tìm nằm cuối, N – M + 2 chuỗi subtext đầu không trùng với pattern).
* Cần nhiều nhất là M(N – M+1) lần so sánh. Vì duyệt tới cuỗi chuỗi nên đầy là trường hợp tìm thấy ở cuối hoặc không tìm thấy và xảy ra trùng mã băm ở tất cả lần so sánh trước đó.
* Cận trên *O*(MN) (vì N > N – M + 1).

Trường hợp trung bình: *O*(N + M) ∈ *O*(M).

Do trong quá trình tìm kiếm có thực hiện tính toán và lưu mã băm nên chí phí bộ nhớ là hằng số *O*(1). Đánh giá:

* + Mặc dù trong trường hơp xấu nhất có độ phức tạp là *O*(MN) không tốt hơn giải thuật Naive, nhưng giải thuật Rabin – Karp hoạt động tốt hơn nhiều trường hợp trung bình và thực tế.
  + Có bước tiền xử lý với độ phức tạp *O*(M) trước khi bắt đầu tìm kiếm.
  + Độ phức tạp *O*(MN). Chi phí bộ nhớ *O*(1).

**Nguồn tham khảo:**

[**https://github.com/doankimhuy-it/String-Matching**](https://github.com/doankimhuy-it/String-Matching)