BÀI TOÁN SO KHỚP CHUỖI (STRING MATCHING)

1. Bài toán mở đầu

Cho 2 chuỗi (string) độ dài và (pattern) độ dài

* Bài toán 1: tìm vị trí xuất hiện đầu tiên của trong .
* Bài toán 2: Viết liên tiếp một số lần chuỗi nhận được chuỗi . Cho chuỗi , tìm chuỗi ngắn nhất. Ví dụ: ,
* Bài toán 3: Đếm số lần xuất hiện chuỗi trong chuỗi và chỉ các vị trí xuất hiện lần lượt.

Bài toán 1: nếu và có kích thước nhỏ thì có thể giải quyết bằng thuật toán Brute Force

* Cho trượt dần trên và so khớp các vị trí với
* Độ phức tạp

Bài toán 2:

Nhận xét:

* Độ dài của chuỗi nhỏ nhất là 1. Ta xét độ dài bắt đầu từ trở đi.
* Điều kiện để chuỗi được lặp lại từ chuỗi là là ước của .

Thuật toán 1

int base(string s) //tìm độ dài của chuỗi P

{

n = s.size();

for (m = 1; m <= n; ++m)

if (n % m == 0 && check(s, m))

return m;

}

0123456789012345

0123012301230123

bool check(string s, int len) //kiểm tra len kí tự đầu của s có được lặp lại

{

for (i = 0; i < s.size(); ++i)

if (s[i] != s[i % len]) return false;

return true;

}

Độ phức tạp:

Thuật toán 2:

Bước 1: lọc ra các ước của n (có thứ tự tăng dần) và lưu vào mảng d

Bước 2: tìm độ dài của chuỗi lặp P

int base(string s) //tìm độ dài của chuỗi P

{

for (i = 0; i < d.size(); ++i)

if (check(s, d[i]))

return d[i];

}

Độ phức tạp , d: số ước của

Bài toán 3: khi 2 chuỗi có độ dài lớn, thuật toán KMP sẽ hoạt động với thời gian tuyến tính (độ phức tạp ).

1. Thuật toán Z

*Định nghĩa hàm :* cho chuỗi có độ dài . Hàm của chuỗi được định nghĩa là dãy số nguyên , trong đó với ý nghĩa là độ dài tiền tố chung dài nhất của 2 chuỗi: và .

Quy ước .

Ví dụ: S="abaabab" có hàm

|  |  |  |  |
| --- | --- | --- | --- |
| i | S[0..n] | S[i..n] | z[i] |
| 0 | abaabab | abaabab | 0 |
| 1 | abaabab | baabab | 0 |
| 2 | abaabab | aabab | 1 |
| 3 | abaabab | abab | 3 |
| 4 | abaabab | bab | 0 |
| 5 | abaabab | ab | 2 |
| 6 | abaabab | b | 0 |

Với dãy số nguyên có được sau khi tính xong hàm , ta có với

* kí tự đầu của chuỗi sẽ xuất hiện lại tại vị trí .
* Số phần tử có cùng giá trị cho biết tần số xuất hiện của kí tự của chuỗi .

Để tìm vị trí xuất hiện của chuỗi trong chuỗi thì ta ghép , với là một kí tự đặc biệt không xuất hiện trong chuỗi . Sau đó tính hàm cho chuỗi mới

*Thuật toán*

Nhận xét:

* Để tính z[i], thuật toán sẽ so sánh các cặp phần tử (s[0], s[i]), (s[1], s[i+1]), (s[2], s[i+2]), …, (s[k-1], s[i+k-1])

Đoạn chương trình

z[i] = 0, với mọi i

for (i = 1; i < n; ++i)

while (i + z[i] < n && s[z[i]] == s[i+z[i]])

++z[i];

Độ phức tạp của thuật toán

Nhận xét cải tiến:

-1

*S*

* Với mỗi vị trí , chuỗi có độ dài phần đầu khớp dài nhất là với chuỗi
* Gọi là vùng thông tin đã được xét bắt đầu từ vị trí .
* Khi tính giá trị , có 2 trường hợp xảy ra:
  + TH1: : vị trí đang xét nằm trong vùng so khớp đã tìm được . 2 chuỗi và là giống nhau. Do đó ta có thể tận dụng giá trị hàm để khởi tạo cho , thay vì luôn khởi tạo với mọi .
  + TH2: : vị trí đang xét nằm trong vùng chưa được so khớp nên không thể tận dụng được những thông tin đã tính được trước đó.

l = r = 0;

for (i = 1; i < n; ++i)

{

if (i <= r) //TH1

z[i] = z[i-l];

while (i + z[i] < n && s[z[i]] == s[i+z[i]])

++z[i];

}

* Lưu ý rằng vì chuỗi bắt đầu từ vị trí có thể mở rộng hơn nữa, trong khi chuỗi bắt đầu từ vị trí có vị trí cuối không được vượt quá , nên ta khởi tạo z[i] = min(z[i-l], r-i+1)
* Ta cố gắng mở rộng đầu dài nhất trong số các đoạn như vậy.

Đoạn chương trình cải tiến

l = r = 0;

for (i = 1; i < n; ++i)

{

if (i <= r)

z[i] = min(z[i-l], r-i+1);

while (i + z[i] < n && s[z[i]] == s[i+z[i]])

++z[i];

if (i + z[i] – 1 > r)

{

l = i;

r = i + z[i] – 1;

}

}

Độ phức tạp của thuật toán cỡ