

# Xử lý xâu

## Xử lý xâu

**Xâu (string)** xuất hiện rất nhiều trong các bài toán. Bài viết này giới thiệu sơ qua một số thuật ngữ cũng như thuật toán về xâu.

## Thuật ngữ

- ▶ Một xâu  $X$  là **xâu con (substring)** của một xâu  $Y$  nếu  $X$  là một chuỗi các ký tự liên tiếp của  $Y$ . Ví dụ:  $ab$  và  $bc$  là 2 xâu con của  $abcd$ . Nhưng  $ac$  thì không phải là xâu con của  $abcd$ .
- ▶ Một xâu  $X$  là **tiền tố (prefix)** của một xâu  $Y$  nếu  $X$  là xâu con của  $Y$  và  $X$  xuất hiện ở đầu của xâu  $Y$ . Ví dụ:  $ab$  là tiền tố của  $abcd$ , nhưng  $bc$  **không** phải là tiền tố của  $abcd$ .  
Một xâu  $X$  là một **tiền tố không tầm thường (proper prefix)** của xâu  $Y$  nếu nó là tiền tố của xâu  $Y$  và khác xâu  $Y$ .
- ▶ Một xâu  $X$  là **hậu tố (suffix)** của một xâu  $Y$  nếu  $X$  là xâu con của  $Y$  và  $X$  xuất hiện ở cuối của xâu  $Y$ . Ví dụ:  $cd$  là hậu tố của  $abcd$ , nhưng  $bc$  **không** phải là hậu tố của  $abcd$ .  
Một xâu  $X$  là một **hậu tố không tầm thường (proper suffix)** của xâu  $Y$  nếu nó là hậu tố của xâu  $Y$  và khác xâu  $Y$ .

## Các dạng bài

### So khớp chuỗi (string matching)

Cho một xâu  $T$  và xâu  $S$ . Tìm tất cả các lần xuất hiện của xâu  $S$  trong xâu  $T$ .

Ví dụ:

```
1 | S = abc
2 | T = abcabcabc
3 |
4 | Các lần xuất hiện: 1, 4, 7.
```

Bài toán này còn được gọi là tìm kiếm **cây kim (needle)** trong **đống rơm (haystack)**, vì nó xuất hiện trong thực tế khi ta cần tìm một xâu rất nhỏ trong một lượng dữ liệu rất lớn (ví dụ Google cần tìm từ khóa trong hàng tỉ trang web).

Có 3 thuật toán chính để giải quyết bài này, đó là:

- [Thuật toán KMP](#)
- [Hash](#)
- [Z Algorithm](#)

## Xâu đối xứng (Palindrome)

Palindrome hay còn gọi là chuỗi đối xứng, chuỗi đối gương là tên gọi của những chuỗi ký tự mà khi viết từ phải qua trái hay từ trái qua phải thì chuỗi đó không thay đổi. VD: MADAM, IOI,...

Có rất nhiều bài tập liên quan đến chuỗi đối xứng. Các bạn có thể tìm đọc ở trong các bài viết:

- [Một vài bài tập QHD về Palindrome](#)
- [Hash](#)
- [Palindrome Tree](#)

## Cấu trúc dữ liệu

- [Trie](#) là CTDL cơ bản nhất trong xử lý chuỗi. Nó giúp giải quyết các bài toán về tìm kiếm chuỗi.
- Lớp CTDL được gọi chung là Suffix Structures gồm:
  - [Suffix Array](#)
  - Suffix Automaton
  - Suffix Tree
  - Aho Corasick

Gọi chung như vậy vì các CTDL này có thể dùng thay thế nhau để giải quyết cùng một lớp bài toán liên quan đến các suffix của cây.

## Các bài Ad-hoc

Trong xử lý chuỗi còn một vài thuật toán chỉ áp dụng được cho 1 bài toán (ad-hoc).


### Thuật toán Manacher

#### Bài toán

Cho chuỗi  $S$ .

- Với mỗi vị trí  $i$  của chuỗi  $S$ , tìm chuỗi đối xứng dài nhất nhận  $i$  là tâm.
- Với mỗi cặp  $i, i + 1$  của chuỗi  $S$ , tìm chuỗi đối xứng dài nhất nhận  $i$  và  $i + 1$  là tâm.

#### Mô tả thuật toán

Tham khảo thêm ở [link](#) 

#### Code

```
const char DUMMY = ' ';
```

```

3  int manacher(string s) {
4      // Để tránh phải xét riêng trường hợp độ dài xâu đối xứng chẵn / lẻ,
5      // ta thêm 1 ký tự DUMMY vào giữa các ký tự của s.
6      // CHÚ Ý: Phải đảm bảo DUMMY không có trong xâu s
7
8      int n = s.size() * 2 - 1;
9      vector <int> f = vector <int>(n, 0);
10
11     // Tạo xâu a bằng cách chèn ký tự DUMMY vào giữa các ký tự của s.
12     // Ví dụ:
13     // s = aabcb
14     // a = a.a.b.c.b
15     string a = string(n, DUMMY);
16     for (int i = 0; i < n; i += 2) a[i] = s[i / 2];
17
18     int l = 0, r = -1, center, res = 0;
19     for (int i = 0, j = 0; i < n; i++) {
20         j = (i > r ? 0 : min(f[l + r - i], r - i)) + 1;
21         while (i - j >= 0 && i + j < n && a[i - j] == a[i + j]) j++;
22         f[i] = --j;
23         if (i + j > r) {
24             r = i + j;
25             l = i - j;
26         }
27
28         int len = (f[i] + i % 2) / 2 * 2 + 1 - i % 2;
29         if (len > res) {
30             res = len;
31             center = i;
32         }
33     }
34     // Với mỗi vị trí i, xâu đối xứng dài nhất nhận i là tâm là [i - f[i], i + f
35     // Ví dụ:
36     // s = aabcb
37     // a = a.a.b.c.b
38     // f = 011010200
39     return res;
40 }

```

## Minimal string rotation




Cho một xâu  $S$ . Xét các xâu thu được từ xâu  $S$  bằng phép xoay. Ví dụ:  $S = abcd$ , thì các xâu thu được từ  $S$  bằng phép xoay là:

- $abcd$
- $bcd a$

- [cdab](#)
- [dabc](#)

Tìm xâu có thứ tự từ điển nhỏ nhất.

### Mô tả thuật toán

Bạn có thể xem [ở đây](#) 

### Code

```

1  // Tính vị trí của xâu xoay vòng có thứ tự từ điển nhỏ nhất của xâu s[]
2  int minmove(string s) {
3      int n = s.length();
4      int x, y, i, j, u, v; // x is the smallest string before string y
5      for (x = 0, y = 1; y < n; ++ y) {
6          i = u = x;
7          j = v = y;
8          while (s[i] == s[j]) {
9              ++ u; ++ v;
10             if (++ i == n) i = 0;
11             if (++ j == n) j = 0;
12             if (i == x) break; // All strings are equal
13         }
14         if (s[i] <= s[j]) y = v;
15         else {
16             x = y;
17             if (u > y) y = u;
18         }
19     }
20     return x;
21 }
```

## Lyndon Decomposition

### Bài toán

**Lyndon word** là các xâu khác rỗng, mà có thứ tự từ điển nhỏ hơn tất cả các xâu thu được bằng phép xoay của nó.

Cho một xâu  $S$ . Tìm cách tách  $S$  thành ít nhất các xâu, sao cho mỗi xâu đều là Lyndon word.

### Code

```

1  void lyndon(string s) {
2      int n = (int) s.length();
3      int i = 0;
4      while (i < n) {
5          int j = i + 1, k = i;
6          while (j < n && s[k] <= s[j]) {
```

```
7         if (s[k] < s[j]) k = i;
8         else ++k;
9         ++j;
10    }
11    while (i <= k) {
12        cout << s.substr(i, j - k) << ' ';
13        i += j - k;
14    }
15 }
16 cout << endl;
17 }
```

Được cung cấp bởi [Wiki.js](#)