

Thuật toán Manacher

Thuật toán Manacher

Người viết: Phạm Hoàng Hiệp - University of Georgia

Review bởi:

- ▶ Nguyễn Minh Hiền - Trường Đại học Công nghệ, ĐHQGHN
- ▶ Nguyễn Minh Nhật - Trường THPT chuyên Khoa học Tự nhiên, ĐHQGHN
- ▶ Ngô Nhật Quang - The University of Texas at Dallas

Giới thiệu

Bài toán

Cho chuỗi S có độ dài n . Hãy tìm tất cả các chuỗi con đối xứng (palindrome) trong chuỗi S . Thuật toán Manacher sẽ xử lý bài toán trên với độ phức tạp thời gian $O(n)$.

Một số khái niệm và ký hiệu trong bài

- ▶ Chuỗi đối xứng (palindrome) là chuỗi mà khi đọc ngược lại thì vẫn là chính nó, ví dụ như *tacocat*, *aaaaaaaa*, 2002
- ▶ Điểm chính giữa của chuỗi đối xứng là vị trí mà khi lật ngược chuỗi đối xứng đó thì vị trí của điểm đó trong chuỗi đó không thay đổi.
 - ▶ Trong chuỗi đối xứng có độ dài lẻ, điểm chính giữa là một chữ cái trong chuỗi. Ví dụ chuỗi *aabaa* có điểm chính giữa là chữ *b*
 - ▶ Trong chuỗi đối xứng có độ dài chẵn, điểm chính giữa là khoảng trống (vị trí giữa của hai chữ cái liên tiếp). Ví dụ chuỗi *aabbaa* có điểm chính giữa là khoảng trống giữa hai chữ *b*
 - ▶ Nếu điểm chính giữa của chuỗi T nào đó là i , ta có thể nói chuỗi T đối xứng qua i
- ▶ S_i là chữ cái thứ i của chuỗi S . $S_{i...j}$ là chuỗi con từ vị trí thứ i đến j của S , bao gồm cả i và j



Thuật toán ngây thơ

Sai lầm thường gặp

Lưu ý, vì có thể có $O(n^2)$ xâu con đối xứng trong một xâu nên có thể dễ dàng lầm tưởng rằng không thể tạo ra thuật toán có độ phức tạp tốt hơn $O(n^2)$ cho bài toán này. Tuy nhiên, với một xâu đối xứng, chúng ta có các xâu con ở phía trong cũng là đối xứng.

Cụ thể, giả sử xâu con $S_{i...j}$ là một xâu đối xứng. Nếu $i + 1 \leq j - 1$ thì ta có $S_{(i+1)...(j-1)}$ cũng là một xâu đối xứng. Ví dụ, xâu $abcdcba$ đối xứng thì có thể dễ dàng thấy được rằng các xâu $bcdcb$, cdc hay d đều đối xứng.

Như vậy, với mỗi điểm chính giữa của một xâu con đối xứng mid , chúng ta có thể lưu lại độ dài dài nhất $d[mid]$ sao cho $S_{(mid-d)...(mid+d)}$ là một xâu đối xứng.

Điểm chính giữa là điểm mà khi đảo ngược trật tự của xâu thì vị trí của điểm này trong xâu không thay đổi. Điểm chính giữa này có thể là một chữ cái hoặc một khoảng trống, tương ứng với xâu đối xứng có độ dài lẻ hoặc chẵn.

Thuật toán ngây thơ

Như vậy, từ quan sát trên, chúng ta có thể ngay lập tức đưa ra thuật toán có độ phức tạp $O(n^2)$. Xét tất cả các vị trí trong xâu (một chữ cái hoặc một khoảng trống), và chạy về hai phía đến khi nào xâu không còn đối xứng nữa.

Cũng như rất nhiều thuật toán liên quan đến xâu khác như Z hay KMP, khi xử lý ở từng vị trí trên xâu, kết quả ở những vị trí trước cho ta rất nhiều dữ liệu để xử lý ở vị trí tiếp theo. Trong bài toán này, chúng ta có thể tận dụng dữ kiện xâu đối xứng như hình vẽ dưới đây.

$$\begin{array}{ccccccc}
 & & & & \text{đối xứng} & & \\
 \dots & S_l & \dots & S_{l+i-d[l+i]} & \dots & S_{l+i} & \dots & S_{l+i+d[l+i]} & \dots & S_{mid} & \dots & S_{r-i-d[l+i]} & \dots & S_{r-i} & \dots & S_{r-i+d[l+i]} & \dots & S_r & \dots \\
 & & & \underbrace{\hspace{10em}}_{\text{đối xứng}} & & & & & & & & \underbrace{\hspace{10em}}_{\text{đối xứng}} & & & & & & &
 \end{array}$$

Giả sử chúng ta có một xâu đối xứng qua mid kéo từ l đến r như hình vẽ trên. Để tính $d[r - i]$ (đã biết hết các giá trị d trước đó), chúng ta có thể dựa vào $S[l + i]$.

Do $l + i$ và $r - i$ đối xứng qua mid nên có thể tận dụng kết quả đã tính với vị trí $l + i$ để tính toán $r - i$ một cách tiết kiệm hơn. Trên đây là ý tưởng chính của thuật toán Manacher. Tiếp theo chúng ta sẽ đi vào chi tiết về thuật toán cũng như cách cài đặt.

Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MaxN = 5e3 + 5;
int D_odd[MaxN];
int D_even[MaxN];
string S;
int N;

void Calc_D_odd() {
    for(int i = 1 ; i <= N ; i++) {
        D_odd[i] = 0;
        while(i - D_odd[i] - 1 > 0 && i + D_odd[i] + 1 <= N && S[i - D_odd[i] - 1] == S[i + D_odd[i] + 1])
            D_odd[i]++;
    }
}

void Calc_D_even() {
    int L = 1;
    int R = 0;
    for(int i = 1 ; i < N ; i++) {
        int j = i + 1;
        D_even[i] = 0;
        while(i - D_even[i] > 0 && j + D_even[i] <= N && S[i - D_even[i]] == S[j + D_even[i]])
            D_even[i]++;
    }
}

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> S;
    N = S.length();
    S = ' ' + S; // for 1-base
    Calc_D_odd();
    Calc_D_even();
    for(int i = 1 ; i < N ; i++) {
        cout << D_odd[i] * 2 + 1 << ' ' << D_even[i] * 2 << ' ';
    }
}
```

```
cout << D_odd[N] * 2 + 1;
}
```

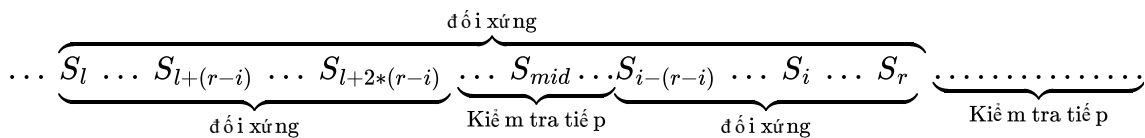
Thuật toán Manacher

Tìm các chuỗi đối xứng có độ dài lẻ

Do chuỗi đối xứng độ dài lẻ có điểm chính giữa là một chữ cái, việc xử lý và tính toán sẽ dễ dàng hơn so với trường hợp chuỗi đối xứng có độ dài chẵn (điểm chính giữa là khoảng trống giữa hai chữ cái). Vì vậy, chúng ta sẽ giải quyết trường hợp này trước.

Chúng ta sẽ tính giá trị d với tất cả các vị trí chứa các ký tự trong chuỗi (tìm các chuỗi đối xứng có độ dài lẻ) theo thứ tự lần lượt từ đầu đến cuối chuỗi với các bước như sau. Chúng ta sẽ đánh số các ký tự của chuỗi S bắt đầu từ 1.

- Đặt $l = 1$ và $r = 0$ (chưa có chuỗi con đối xứng nào)
- Duy trì giá trị l và r thỏa mãn chuỗi con từ l đến r là chuỗi đối xứng với r lớn nhất đã được tìm thấy. Chúng ta luôn có $(l + r)/2 < i$ (i nằm ở phía bên phải của chuỗi đối xứng)
- Tại điểm i ,
 - nếu $i > r$, thì chạy trâu để tìm $d[i]$ như trong thuật toán có độ phức tạp $O(n^2)$
 - nếu $i \leq r$, ta lấy giá trị $d[l + (r - i)]$ (điểm đối xứng với i qua mid). Tại đây, trong phần lớn trường hợp, chúng ta có thể gán luôn $d[i] = d[l + (r - i)]$. Tuy vậy, vẫn có trường hợp đặc biệt sau đây cần lưu ý (Xem hình vẽ dưới)



- Nếu chuỗi đối xứng dài nhất ở vị trí $l + r - i$ không kéo dài đến l , thì chuỗi đối xứng dài nhất tương ứng qua mid ở vị trí i cũng không kéo dài đến r .
- Từ đó ta có thể khẳng định, nếu tiếp tục mở rộng ra thì phạm vi của hai chuỗi này vẫn nằm trong chuỗi đối xứng to kéo từ l đến r .
- Do vậy, khi tiếp tục mở rộng về hai phía thì chuỗi con được mở rộng ở vị trí i vẫn sẽ giống với $(l + (r - i))$ và không còn đối xứng nữa.
- Tuy nhiên, nếu chuỗi đối xứng dài nhất ở $(l + (r - i))$ kéo dài quá l , chúng ta chỉ có thể tạm thời xét $d[i] = r - i$ do không thể đảm bảo các chữ ở phía ngoài giống nhau.
- Tại thời điểm này, chúng ta chạy trâu tiếp từ vị trí r để tìm được $d[i]$.
- Cập nhật lại l và r nếu cần thiết.

Tìm các chuỗi đối xứng có độ dài chẵn



một ký tự). Tuy nhiên, chi tiết cài đặt cần cẩn thận để có được kết quả chính xác.

Để tránh phải xử lý cẩn thận trong trường hợp chuỗi đối xứng độ dài chẵn, bạn đọc có thể cân nhắc sử dụng cách làm như sau.


Thêm các ký tự đặc biệt giữa hai ký tự liên tiếp trong xâu, khi đó các xâu đối xứng độ dài chẵn sẽ là các xâu đối xứng độ dài lẻ với điểm chính giữa là một ký tự đặc biệt. Các xâu đối xứng độ dài lẻ vẫn là các xâu đối xứng độ dài lẻ với điểm chính giữa là các chữ cái ban đầu trong xâu.

Ký tự đặc biệt được thêm cần phải giống nhau (để đảm bảo tính đối xứng) và khác với tất cả các ký tự được dùng trong xâu. Ví dụ, *mike4235* có thể chuyển thành *m.i.k.e.4.2.3.5*

Độ phức tạp

Ở mỗi lần tính $d[i]$, nếu $d[i] + i < r$ thì chúng ta chỉ mất $O(1)$. Tuy nhiên, nếu $d[i] + i > r$ thì chúng ta sẽ đặt lại giá trị của r thành $d[i] + i$. Do r không giảm và $r \leq n$ nên tổng độ phức tạp không vượt quá $O(n)$.

Cài đặt

Dưới đây là code mẫu của thuật toán Manacher cho bài toán được nêu trong phần **Giới thiệu**. Các bạn có thể tự cài đặt và nộp ở link sau: [Library checker - Enumerate Palindromes](#) 

Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MaxN = 5e5 + 5;
int D_odd[MaxN];
int D_even[MaxN];
string S;
int N;

void Calc_D_odd() {
    int L = 1;
    int R = 0;
    for(int i = 1 ; i <= N ; i++) {
        if(i > R) D_odd[i] = 0;
        else D_odd[i] = min(R - i, D_odd[L + (R - i)]);
        while(i - D_odd[i] - 1 > 0 && i + D_odd[i] + 1 <= N && S[i - D_odd[i] - 1] == S[i + D_odd[i] + 1])
            D_odd[i]++;
    }

    if(i + D_odd[i] > R) {
        R = i + D_odd[i];
        L = i - D_odd[i];
    }
}

void Calc_D_even() {
    int L = 1;
    int R = 0;
    for(int i = 1 ; i < N ; i++) {
        int j = i + 1;
        if(j > R) D_even[i] = 0;
        else D_even[i] = min(R - j + 1, D_even[L + (R - j)]);
    }
}
```

```

34     while(i - D_even[i] > 0 && j + D_even[i] <= N && S[i - D_even[i]] == S[j +
35         D_even[i]++);
36     }
37     if(i + D_even[i] > R) {
38         R = i + D_even[i];
39         L = j - D_even[i];
40     }
41 }
42 }
43
44 signed main() {
45     ios_base::sync_with_stdio(0);
46     cin.tie(0); cout.tie(0);
47     cin >> S;
48     N = S.length();
49     S = ' ' + S; // for 1-base
50     Calc_D_odd();
51     Calc_D_even();
52     for(int i = 1 ; i < N ; i++) {
53         cout << D_odd[i] * 2 + 1 << ' ' << D_even[i] * 2 << ' ';
54     }
55     cout << D_odd[N] * 2 + 1;
56 }

```

Ví dụ

CF Sonya and Matrix Beauty [🔗](#)

Tóm tắt đề bài

Cho hai số nguyên dương $n, m \leq 250$ và một ma trận kích thước $n * m$ gồm các chữ cái thường trong bảng chữ cái tiếng Anh.

Một ma trận con (i_1, j_1, i_2, j_2) ($1 \leq i_1 \leq i_2 \leq n, 1 \leq j_1 \leq j_2 \leq m$) là ma trận gồm các phần tử a_{ij} của ma trận ban đầu sao cho $i_1 \leq i \leq i_2$ và $j_1 \leq j \leq j_2$.

Một ma trận con được định nghĩa là **đẹp** nếu có cách để thay đổi thứ tự các chữ cái trên các hàng (không thay đổi thứ tự trên các cột) sao cho tất cả các hàng và cột của ma trận đều con đó đều là xâu đối xứng.

Xác định số ma trận con **đẹp** trong ma trận đã cho.

Gợi ý

- Từ điều kiện đề bài, thử tìm cách sắp xếp một ma trận con và xác định điều kiện để tạo ra xâu đối xứng trên hàng và cột.
- Do các chữ cái trên một hàng chỉ thay đổi thứ tự, khi nào thì hai hàng được coi là "bằng nhau"?

Lời giải

- Trước hết, chúng ta có nhận xét rằng để ma trận con thỏa mãn điều kiện thì các hàng phải sắp xếp lại được thành xâu đối xứng (có 0 hoặc 1 chữ cái có số lần xuất hiện lẻ).
- Để cả ma trận con đối xứng thì hai hàng đối xứng qua điểm đối xứng *mid* phải có số lần xuất hiện bằng nhau với tất cả các chữ cái.
- Như vậy, khi đã có cách so sánh hai hàng "bằng nhau", chúng ta sẽ có thể đếm số ma trận con thỏa mãn bằng cách cố định hai cột bất kỳ và đếm số xâu đối xứng.
- Khi đó, chúng ta có thể sử dụng thuật toán Manacher trên các hàng.
- Lưu ý, cần đảm bảo các điều kiện về hàng có thể đối xứng.
- Như vậy độ phức tạp là $O(n^3 * 26)$, trong đó $O(n^2)$ là số cặp cột, $O(n)$ cho thuật toán Manacher và $O(26)$ để so sánh giữa 2 hàng (26 là kích thước bảng chữ cái).

Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
const int MaxN = 255;
int Cnt[MaxN][26];
int Cnt_Odd[MaxN];
int D_odd[MaxN];
int D_even[MaxN];
char c[MaxN][MaxN];
bool Ok[MaxN];
int N,M;
int Ans = 0;
bool Equal(int Row1, int Row2) {
    if(!Ok[Row1]) return false;
    if(!Ok[Row2]) return false;
    for(int j = 0 ; j < 26 ; j++) {
        if(Cnt[Row1][j] != Cnt[Row2][j]) return false;
    }
    return true;
}
void Calc_D_odd() {
    int L = 1;
    int R = 0;
    for(int i = 1 ; i <= N ; i++) {
        if(i > R) D_odd[i] = 0;
        else D_odd[i] = min(R - i, D_odd[L + (R - i)]);
        if (Ok[i]) while(i - D_odd[i] - 1 > 0 && i + D_odd[i] + 1 <= N && Equal(i - D_odd[i], i + D_odd[i]))
            D_odd[i]++;
        Ans += (D_odd[i] + Ok[i]);
        if(i + D_odd[i] > R) {
            R = i + D_odd[i];
            L = i - D_odd[i];
        }
    }
}
```

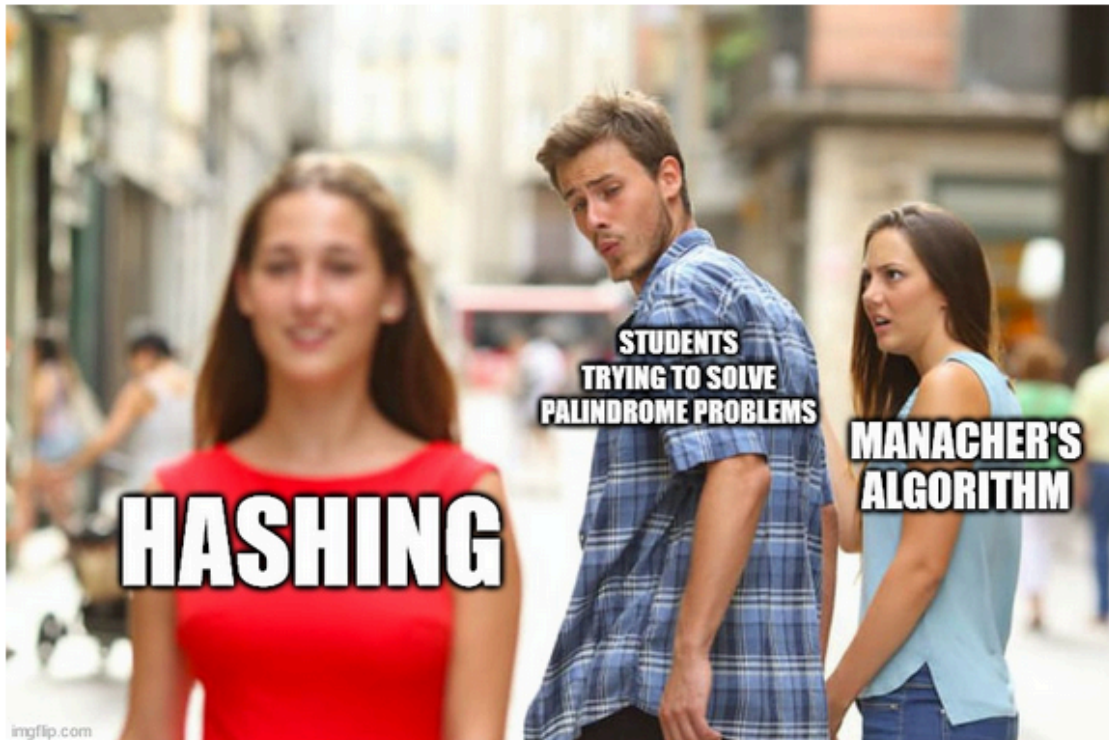
```

37 void Calc_D_even() {
38     // D_even[i] is the value for the space between i and i + 1
39     int L = 1;
40     int R = 0;
41     for(int i = 1 ; i < N ; i++) {
42         int j = i + 1;
43         if(j > R) D_even[i] = 0;
44         else D_even[i] = min(R - j + 1, D_even[L + (R - j)]);
45         while(i - D_even[i] > 0 && j + D_even[i] <= N && Equal(i - D_even[i], j +
46             D_even[i]++);
47     }
48     Ans += D_even[i];
49     if(i + D_even[i] > R) {
50         R = i + D_even[i];
51         L = j - D_even[i];
52     }
53 }
54 }
55 signed main() {
56     ios_base::sync_with_stdio(0);
57     cin.tie(0); cout.tie(0);
58     cin >> N >> M;
59     for(int i = 1 ; i <= N ; i++) {
60         cin >> c[i] + 1;
61     }
62     for(int c1 = 1 ; c1 <= M ; c1++) {
63         for(int c2 = c1 ; c2 <= M ; c2++) {
64             bool ok = true;
65             for(int i = 1 ; i <= N ; i++) {
66                 int t = c[i][c2] - 'a';
67                 Cnt[i][t]++;
68
69                 if(Cnt[i][t] & 1) Cnt_Odd[i]++;
70                 else Cnt_Odd[i]--;
71
72                 if(Cnt_Odd[i] > ((c2 - c1 + 1) & 1)) Ok[i] = false;
73                 else Ok[i] = true;
74             }
75             Calc_D_odd();
76             Calc_D_even();
77         }
78         for(int i = 1 ; i <= N ; i++) {
79             for(int j = 0 ; j < 26 ; j++) {
80                 Cnt[i][j] = 0;
81             }
82             Cnt_Odd[i] = 0;
83         }
84     }
85     cout << Ans;
86 }

```


Nhận xét về thuật toán

- ▶ Thuật toán Manacher là thuật toán có ý tưởng khá đơn giản: tận dụng các dữ liệu có sẵn để giảm độ phức tạp khi tính toán trên xâu.
- ▶ Chúng ta đã từng gặp ý tưởng tương tự khi tính toán [Z-function](#) hay [prefix function](#).
- ▶ Thuật toán Manacher có thể được sử dụng trong hơi ít các bài toán. Tuy nhiên, thuật toán có ý tưởng tự nhiên và dễ cài đặt nên có thể có ích khi thi đấu.



Bài tập luyện tập

- ▶ [CSES Longest palindrome](#)
- ▶ [UVA Extend to Palindrome](#)
- ▶ [Gym QueryreuQ](#)
- ▶ [CF Prefix-Suffix Palindrome](#)
- ▶ [SPOJ Number of Palindromes](#)
- ▶ [Kattis Palindromes](#)

Được cung cấp bởi [Wiki.js](#)