

Mảng hậu tố (Suffix Array) và một số ứng dụng

MỤC LỤC

MỤC LỤC	1
A. MỞ ĐẦU.....	2
B. NỘI DUNG.....	3
1. Bài toán mở đầu: Đếm xâu con phân biệt.....	3
2. Mảng hậu tố	3
2.1. Khái niệm	3
2.2. Xây dựng mảng hậu tố	4
2.3. Tiền tố chung dài nhất của hai hậu tố	6
2.4. Mảng tiền tố chung dài nhất	8
3. Một số bài tập áp dụng.....	11
3.1. Kiểm tra xâu con	11
3.2. Đếm số lượng xâu con trong xâu	13
3.3. Đếm số lượng xâu con phân biệt độ dài K	15
3.4. Xâu con chung dài nhất.....	17
3.5.Tìm xâu con nhỏ thứ K	18
C. KẾT LUẬN	21
D. TÀI LIỆU THAM KHẢO.....	21

A. MỞ ĐẦU

Các bài toán xử lí xâu thường có mặt trong các kì thi học sinh giỏi Quốc gia, Khu vực và Quốc tế bởi nó có tính ứng dụng thực tiễn cao. Mảng hậu tố (Suffix Array) là một trong những phương pháp tiếp cận để giải lớp những bài toán này một cách chính xác và hiệu quả cao. Bên cạnh đó, cài đặt mảng hậu tố ngắn gọn và đơn giản hơn nhiều so với việc cài đặt cây hậu tố (Suffix Tree).

Trong phạm vi chuyên đề này, tôi xin được trình bày những hiểu biết và kinh nghiệm thực tế của bản thân trong quá trình bồi dưỡng đội tuyển dự thi học sinh giỏi cấp Quốc gia ở địa phương. Những bài toán và lời giải được lựa chọn hi vọng có thể giúp ích được cho những thầy cô có cùng quan tâm đến phương pháp tiếp cận này.

Nhân đây, tôi xin gửi lời cảm ơn quý thầy cô, các bạn đồng nghiệp đã chia sẻ tài liệu và góp ý bổ sung để tôi hoàn thiện chuyên đề này. Dù đã rất cố gắng nhưng không thể tránh khỏi những thiếu sót, rất mong nhận được những góp ý từ quý thầy cô.

B. NỘI DUNG

1. Bài toán mở đầu: Đếm xâu con phân biệt

Cho xâu kí tự S , xâu con của S là một đoạn gồm các kí tự liên tục của S . Ví dụ các xâu $I, IO, O \dots$ là các xâu con của xâu IOI . Hai xâu không bằng nhau được gọi là hai xâu phân biệt.

Yêu cầu: Đếm số lượng xâu con phân biệt khác rỗng của S .

Dữ liệu vào: từ tệp văn bản **DEMXAUPB.INP** ghi duy nhất xâu S gồm các chữ cái in hoa.

Kết quả: ghi ra tệp văn bản **DEMXAUPB.OUT** gồm một dòng ghi một số là số lượng các xâu con phân biệt của S .

Ví dụ:

DEMXAUPB.INP	DEMXAUPB.OUT
IOI	5

Ràng buộc:

- Có 20 % test độ dài của $S \leq 200$ tương ứng 20 % số điểm;
- Có 40 % test độ dài của $S \leq 1.000$ tương ứng 40 % số điểm;
- Có 40 % test độ dài của $S \leq 100.000$ tương ứng 40 % số điểm.

Bộ Test có thể download tại đây:

https://drive.google.com/drive/folders/1L31HkQGyZqK_boloSVIu8brgfL3RpvA?usp=sharing

Subtest đầu tiên có thể dùng thuật toán duyệt với độ phức tạp $O(n^3)$, subtest thứ hai có thể lấy ra tất cả các xâu con có cùng độ dài, đẩy các xâu đó vào *set* để loại bỏ các xâu trùng, nếu sử dụng Hash (<http://vnoi.info/wiki/algo/string/hash>) để mã hóa xâu thì tính toán là $O(n^2 \log n)$. Subtest thứ ba có thể giải quyết với chi phí $O(n \log^2 n)$ bằng mảng hậu tố.

2. Mảng hậu tố

2.1. Khái niệm

Cho xâu S có độ dài N , các kí tự được đánh chỉ số từ $0..N - 1$. Hậu tố thứ i của xâu S là xâu con $S[i..N - 1]$ ($0 \leq i < N$). Ví dụ $S = 'abaab'$ có các hậu tố là:

Thứ tự	Hậu tố
0	<i>abaab</i>
1	<i>baab</i>
2	<i>aab</i>
3	<i>ab</i>
4	<i>b</i>

Mảng hậu tố (*SA*) của một xâu *S* là mảng chứa các chỉ số bắt đầu của các hậu tố của *S* được sắp xếp theo thứ tăng dần của các hậu tố. Ví dụ $S = 'abaab'$ có các hậu tố được sắp xếp tăng dần là:

Thứ tự	Hậu tố
2	<i>aab</i>
3	<i>ab</i>
0	<i>abaab</i>
4	<i>b</i>
1	<i>ba</i>

Do đó, *SA* của xâu *S* là [2,3,0,4,1].

2.2. Xây dựng mảng hậu tố

Nếu tạo ra tất cả các xâu hậu tố của *S* rồi sắp lại bằng *quicksort* hoặc *sort* (*C⁺⁺*) để có *SA* thì chi phí là $O(N^2 \log N)$. Sau đây là một trong những phương pháp tạo ra *SA* với chi phí $O(N \log^2 N)$.

Tư tưởng của thuật toán này là lưu lại thứ tự của các hậu tố khi sắp xếp tăng dần theo 2^k ($k = 0..[\log N]$) kí tự đầu tiên (tiền tố độ dài 2^k) của chúng. Các thứ tự này được tính ở bước thứ k . Ta sử dụng mảng *P* kích thước $[\log N] \times N$ để lưu lại tất cả các giá trị này. Gọi A_i^k là tiền tố độ dài 2^k của hậu tố i , khi đó vị trí của A_i^k sau khi sắp xếp tăng dần là $P[k][i]$, $k = 0..[\log N], i = 0..N$. Việc lưu trữ này giúp cho việc so sánh hai xâu có độ dài 2^{k+1} chỉ có chi phí $O(1)$. Việc so sánh này được thực hiện như sau: Nếu $P[k][i] < P[k][j]$ thì $P[k+1][i] < P[k+1][j]$, điều này cũng tương tự nếu $P[k][i] > P[k][j]$ thì $P[k+1][i] > P[k+1][j]$. Nếu $P[k][i] = P[k][j]$ thì ta so sánh nửa sau của hai hậu tố là $P[k][i + 2^k]$ và $P[k][j + 2^k]$ như nửa đầu. Nếu nửa sau cũng bằng nhau thì hai xâu đó có cùng thứ hạng.

Như vậy, *SA* sẽ có được sau khi thực hiện $\log N$ lần lặp (do $k = 0..[\log N]$), tại mỗi lần lặp, ta sắp xếp lại các tiền tố độ dài 2^k của các N hậu tố bằng hàm *sort* với chi phí $O(N \log N)$ nên chi phí cho toàn bộ thuật toán là $O(N \log^2 N)$.

Để minh họa cho cách cài đặt của thuật toán này, ta xét bài toán sau: Cho xâu *S* gồm các chữ cái in thường có độ dài không quá 10^5 kí tự. Hãy tìm mảng hậu tố tương ứng của xâu *S*.

Ví dụ:

Input	Output
<i>abaab</i>	2 3 0 4 1

Chương trình:

```
#include <bits/stdc++.h>
const int maxN = 1e5 + 7; // Độ dài tối đa của xâu S
const int maxK = 20; // giá trị lớn nhất của logN
using namespace std;

int SA[maxN]; // SA[i] lưu chỉ số của hậu tố xếp hạng thứ i sau khi được sắp xếp tăng
int P[maxK][maxN]; // P[k][i] là thứ hạng của hậu tố thứ i khi sắp tăng theo tiền tố độ
//dài 2^k
int _2Pow[maxK]; // _2Pow[i] lưu giá trị 2^i

struct Suffix { // Lưu thông tin của một hậu tố tại mỗi bước k
    int Pos; // Lưu vị trí của hậu tố
    int Ft; // Lưu thứ hạng tiền tố độ dài 2^{k-1}
    int Nt; // Lưu thứ hạng của 2^{k-1} kí tự tiếp theo đứng ngay sau tiền tố độ dài 2^{k-1}
    bool operator < (Suffix const &X) const { // Phép toán so sánh bé hơn
        if (Ft != X.Ft) return Ft < X.Ft;
        return Nt < X.Nt;
    }
    bool operator ==(Suffix const &X) const { // Phép toán so sánh bằng
        if (Ft != X.Ft) return false;
        return Nt == X.Nt;
    }
} L[maxN];

int main()
{
    for (int k=0; k<maxK; ++k) _2Pow[k] = 1 <<k;
    int N; string S; cin>>S; N = S.length();

    for (int i=0; i<N; ++i) P[0][i] = S[i] - 'a';

    for (int k=1; _2Pow[k-1]<N; ++k) {
        for (int i=0; i<N; ++i) {
            L[i].Ft = P[k-1][i];
            L[i].Nt = i+_2Pow[k-1]<N ? P[k-1][i+_2Pow[k-1]] : -1;
            L[i].Pos = i;
        }
        sort(L, L+N);
        for (int i=0; i<N; ++i) { // Tính lại thứ hạng ở bước thứ k
            P[k][L[i].Pos] = i>0 && L[i-1] == L[i] ? P[k][L[i-1].Pos] : i;
        }
    }
    for (int i=0; i< N; ++i) SA[i] = L[i].Pos;
    for (int i=0; i< N; ++i) cout<<SA[i]<<" ";
    return 0;
}
```

Bài này có thể nộp tại đây (để ACC chỉ cần thêm kí tự '#' ở cuối xâu):
<https://codeforces.com/edu/course/2/lesson/2/1/practice/contest/269100/problem/A>

Ngoài cách này ra, nếu ta dùng Radix Sort để sắp xếp các hậu tố của xâu S thì chi phí thuật toán tạo SA là $O(N \log N)$ nhưng khó cài đặt và thời gian giảm đi không đáng kể nên không có hiệu quả cao trong các kì thi.

2.3. Tiết tố chung dài nhất của hai hậu tố

Ta xét bài toán sau: Cho hai xâu S và T gồm các chữ cái in thường, mỗi xâu có không quá 100000 kí tự. Tìm các vị trí xâu T xuất hiện trong xâu S . Ví dụ:

Input	Output
$abaab$	
ab	0 3

Ta thấy nếu xâu T xuất hiện tại vị trí i của xâu S thì T là một tiền tố của hậu tố thứ i ở trong S . Để dễ dàng kiểm tra điều kiện này ta nối xâu T và sau xâu S như sau: $ST = S + "#" + T$, xâu ST mới là $abaab#ab$ và nó có các hậu tố là:

Thứ tự	Hậu tố của ST
0	$abaab#ab$
1	$baab#ab$
2	$aab#av$
3	$ab#ab$
4	$b#ab$
5	$#ab$
6	ab
7	b

Ta thấy các hậu tố từ 0 đến 4 tương ứng các hậu tố xâu S , các hậu tố 6, 7 là của xâu T . Như vậy, các vị trí xuất hiện xâu T trong xâu S ban đầu là những vị trí có tiền tố chung dài nhất của hậu tố với xâu T (trong ví dụ này là hậu tố thứ 6) với các hậu tố trong ST bằng với độ dài xâu T .

Tiết tố chung dài nhất của hai hậu tố i và j ($lcp(i, j)$) được tính dựa vào mảng $P[maxK][maxN]$ với chi phí $O(\log N)$ như sau:

```
int lcp(int i, int j) {
    if (i==j) return N-i+1;
    int ans = 0;
    for (int k=maxK-1; k>=0; --k) {
```

```

        if (i+_2Pow[k]<=N && j+_2Pow[k]<=N &&
            P[k][i] == P[k][j]) {
            ans += _2Pow[k];
            i += _2Pow[k];
            j += _2Pow[k];
        }
    }
    return ans;
}

```

Sau đây là chương trình giải quyết bài toán trên:

```

#include <bits/stdc++.h>
const int maxN = 2e5 + 7;
const int maxK = 20;

using namespace std;

int SA[maxN], P[maxK][maxN], _2Pow[maxK], N;

struct Suffix {
    int Pos, Ft, Nt;
    bool operator < (const Suffix &X) const {
        if (Ft != X.Ft) return Ft < X.Ft;
        return Nt < X.Nt;
    }
    bool operator == (const Suffix &X) const {
        if (Ft != X.Ft) return false;
        return Nt == X.Nt;
    }
} L[maxN];

int lcp(int i,int j) {
    if (i==j) return N-i+1;
    int ans = 0;
    for (int k=maxK-1; k>=0; --k) {
        if (i+_2Pow[k]<=N && j+_2Pow[k]<=N &&
            P[k][i] == P[k][j]) {
            ans += _2Pow[k];
            i += _2Pow[k];
            j += _2Pow[k];
        }
    }
    return ans;
}

int main()
{
    string S,T; cin>>S>>T;
    int nS = S.length();
    int nT = T.length();
    string ST = S + "#" + T;
}

```

```

N = ST.size();

for (int k=0; k<maxK; ++k) _2Pow[k] = 1 <<k;

for (int i=0; i<N; ++i) P[0][i] = ST[i] - 'a';

for (int k=1; _2Pow[k-1] < N; ++k) {
    for (int i=0; i<N; ++i) {
        L[i].Ft = P[k-1][i];
        L[i].Nt = i+_2Pow[k-1]<N ?
                    P[k-1][i+_2Pow[k-1]]:-1;
        L[i].Pos = i;
    }
    sort(L, L+N);
    for (int i=0; i<N; ++i) {
        P[k][L[i].Pos] = i>0 && L[i]==L[i-1] ?
                           P[k][L[i-1].Pos]:i;
    }
}
for (int i=0; i<N; ++i) SA[i] = L[i].Pos;

for (int i=0; i<nS; ++i) {
    if (lcp(i, nS+1) == nT) cout<<i<<" ";
}

return 0;
}

```

Bài này có thể nộp tại đây: <https://www.hackerearth.com/practice/data-structures/advanced-data-structures/suffix-arrays/tutorial/>

2.4. Mảng tiền tố chung dài nhất

Khi các hậu tố của xâu S được sắp xếp theo thứ tự tăng dần thì tiền tố chung dài nhất của hai hậu tố đứng liền nhau chính là tiền tố chung dài nhất của hậu tố đó với các hậu tố khác trong xâu.

Gọi $LCP[i]$ là tiền tố chung dài nhất của hậu tố xếp hạng thứ i và hậu tố xếp hạng thứ $i + 1$ sau khi đã sắp xếp tăng, khi đó số xâu con của xâu S bắt đầu tại vị trí $SA[i]$ chính là độ dài của hậu tố thứ $SA[i]$, số xâu con do hậu tố này tạo ra bị trùng với các xâu con được tạo ra từ các vị trí khác chính là $LCP[i]$. Do vậy số xâu con phân biệt của xâu S có độ dài N chính bằng hiệu của tổng độ dài các hậu tố và tổng $LCP[i]$ hay chính bằng $\frac{n(n+1)}{2} - \sum_{i=0}^{n-1} LCP[i]$. Ví dụ xét $S = 'abaab'$ ta có:

i	Hậu tố	$SA[i]$	$LCP[i]$
0	$abaab$	2	1
1	$baab$	3	2
2	aab	0	0
3	ab	4	1
4	b	1	0

Tổng số xâu con là: $\frac{5 \times 6}{2} = 15$, tổng số xâu con bị trùng là: $1 + 2 + 0 + 1 + 0 = 4$. Do vậy số lượng các xâu con phân biệt là $15 - 4 = 11$.

Để tính các giá trị của $LCP[i]$ ta có thể sử dụng hàm $lcp(i, j)$ ở trên với chi phí tính toán là $O(N \log N)$ như sau:

```
for (int i=0; i<N-1; ++i) {
    LCP[i]=lcp(SA[i], SA[i+1]);
}
```

Ngoài ra ta có thể sử dụng thuật toán Kasai để tính mảng LCP với chi phí $O(N)$ như sau:

```
for (int i=0; i<N; ++i) Pos[SA[i]] = i, LCP[i] = 0;
int k = 0;
for (int i=0; i<N; ++i) {
    if (Pos[i] == N-1) {
        k = 0; continue;
    }
    int j = SA[Pos[i]+1];
    while (i+k < N && j+k < N && S[i+k]==S[j+k]) ++k;
    LCP[Pos[i]] = k;
    if (k) --k;
}
```

Thuật toán trên không dùng đến mảng $P[maxK][maxN]$ nên ta có thể giảm kích thước mảng này bằng cách chỉ cần lưu hai dòng cuối cùng của mảng P . Chương trình giải quyết bài toán đếm số lượng xâu con phân biệt ở phần bài toán mở đầu được viết lại như sau:

```
#include <bits/stdc++.h>
using namespace std;
const int maxN = 4e5 + 7;
const int maxK = 20;

struct Suffix {
    int Pos, Ft, Nt;
    bool operator < (const Suffix &X) const {
        if (Ft != X.Ft) return Ft < X.Ft;
        return Nt < X.Nt;
    }
} L[maxN];
```

```

int SA[maxN], LCP[maxN], _2Pow[maxK], P[2][maxN], Pos[maxN];
int N; string S;

int main()
{
    for (int i=0; i<maxK; ++i) _2Pow[i] = 1<<i;

    cin>>S; N = S.size();

    for (int i=0; i<N; ++i) P[0][i] = S[i] - 'a';
    int last = 0;

    for (int k=1; _2Pow[k-1]<N; ++k) {
        int now = 1 - last;
        for (int i=0; i<N; ++i) {
            L[i].Ft = P[last][i];
            L[i].Nt = i+_2Pow[k-1]<N ?P[last][i+_2Pow[k-1]]: -1;
            L[i].Pos = i;
        }
        sort(L, L+N);
        for (int i=0; i<N; ++i){
            P[now][L[i].Pos] = i>0 && L[i].Ft == L[i-1].Ft
                && L[i].Nt == L[i-1].Nt ?
                    P[now][L[i-1].Pos]:i;
        }
        last = now;
    }

    for (int i=0; i<N; ++i) SA[i] = L[i].Pos;

    for (int i=0; i<N; ++i) Pos[SA[i]] = i, LCP[i] = 0;
    int k = 0;
    for (int i=0; i<N; ++i) {
        if (Pos[i] == N-1) {
            k = 0; continue;
        }
        int j = SA[Pos[i]+1];
        while (i+k <N && j+k <N && S[i+k]==S[j+k]) ++k;
        LCP[Pos[i]] = k;
        if (k) --k;
    }
    long long ans = N;
    ans = ans*(ans+1)/2;
    for (int i=0; i<N; ++i) ans -= LCP[i];
    cout<<ans;
    return 0;
}

```

Bài này có thể nộp chấm trực tuyến tại địa chỉ sau đây:

<https://codeforces.com/edu/course/2/lesson/2/5/practice/contest/269656/problem/A>

3. Một số bài tập áp dụng

Sau đây tôi xin giới thiệu một số bài toán, hướng dẫn giải và code mẫu sử dụng mảng hậu tố. Ngoài cách sử dụng mảng hậu tố có thể có nhiều cách giải khác mang lại hiệu quả cao hơn xin phép không trình bày ở đây.

3.1. Kiểm tra xâu con

Cho xâu S và n truy vấn, truy vấn thứ i là xâu T_i . Với mỗi truy vấn, kiểm tra xem xâu T_i có xuất hiện trong xâu S hay không?

Dữ liệu vào: Dòng đầu ghi xâu S có độ dài không vượt quá 3×10^5 kí tự. Dòng thứ hai ghi số nguyên dương n ($1 \leq n \leq 3 \times 10^5$). Dòng thứ i trong n dòng tiếp theo ghi xâu T_i , tổng độ dài các xâu T_i không vượt quá 3×10^5 . Các kí tự trong các xâu đều là chữ cái in thường.

Kết quả: Mỗi truy vấn ghi trên một dòng, nếu T_i xuất hiện trong S thì ghi "Yes", ngược lại ghi "No".

Ví dụ:

Input	Output
ababba	Yes
3	No
ba	Yes
baba	
abba	

Link chấm: <https://codeforces.com/edu/course/2/lesson/2/3/practice/contest/269118/problem/A>

❖ Hướng dẫn:

Tạo SA của xâu S . Với mỗi truy vấn T_i ta thực hiện tìm kiếm nhị phân trong SA , nếu tồn tại m ($0 \leq m < N$) sao cho T_i là một tiền tố của hậu tố $SA[m]$ thì T_i có xuất hiện trong S .

❖ Code mẫu:

```
#include <bits/stdc++.h>
using namespace std;
const int maxN = 3e5 + 7;
const int maxK = 20;

int N; string S;

struct Suffix {
    int Pos, Ft, Nt;
    bool operator < (const Suffix &X) const {
        if (Ft != X.Ft) return Ft < X.Ft;
        else return Nt < X.Nt;
    }
};
```

```

        return Nt < X.Nt;
    }
} L[maxN];

int P[2][maxN], _2Pow[maxK];
int SA[maxN];

int main()
{
    cin>>S; N = S.length();
    for (int i=0; i<20; ++i) _2Pow[i] = 1<<i;

    for (int i=0; i<N; ++i) P[0][i] = S[i] - 'a';

    int last = 0;
    for (int k=1; _2Pow[k]<N; ++k) {
        int now = 1 - last;
        for (int i=0; i<N; ++i) {
            L[i].Ft = P[last][i];
            L[i].Nt = i + _2Pow[k-1] < N ?
                P[last][i + _2Pow[k-1]] : -1;
            L[i].Pos = i;
        }
        sort(L, L+N);
        for (int i=0; i<N; ++i) {
            P[now][L[i].Pos] = i>0 && L[i].Ft == L[i-1].Ft &&
                L[i].Nt == L[i-1].Nt ? P[now][L[i-1].Pos] : i;
        }
        last = now;
    }
    for (int i=0; i<N; ++i) SA[i] = L[i].Pos;

    int test; cin>>test;
    while (test--) {
        string T; cin>>T;
        int nt = T.size();
        int l=0, r = N-1;
        while (l<=r) {
            int m = (l+r)/2;
            string X = S.substr(SA[m], nt);
            if (X<T) l = m + 1;
            else if (X>T) r = m - 1;
            else break;
        }
        if (l<=r) cout<<"Yes\n"; else cout<<"No\n";
    }
    return 0;
}

```

3.2. Đếm số lượng xâu con trong xâu

Cho xâu S và n truy vấn, truy vấn thứ i là xâu T_i . Với mỗi truy vấn, kiểm tra xem xâu T_i có xuất hiện bao nhiêu lần trong xâu S ?

Dữ liệu vào: Dòng đầu ghi xâu S có độ dài không vượt quá 3×10^5 kí tự. Dòng thứ hai ghi số nguyên dương n ($1 \leq n \leq 3 \times 10^5$). Dòng thứ i trong n dòng tiếp theo ghi xâu T_i , tổng độ dài các xâu T_i không vượt quá 3×10^5 . Các kí tự trong các xâu đều là chữ cái in thường.

Kết quả: Mỗi truy vấn ghi trên một dòng là số lần T_i xuất hiện trong S .

Ví dụ:

Input	Output
ababba	2
3	0
ba	1
baba	
abba	

Link chấm: <https://codeforces.com/edu/course/2/lesson/2/3/practice/contest/269118/problem/B>

❖ *Hướng dẫn:*

Tạo SA của xâu S . Với mỗi truy vấn T_i ta thực hiện tìm kiếm nhị phân trong SA , vị trí nhỏ nhất l và lớn nhất r ($0 \leq l \leq r < N$) sao cho T_i là một tiền tố của hậu tố $SA[l]$ và $SA[r]$. Khi đó số lần xuất hiện là $r - l + 1$.

❖ *Code mẫu:*

```
#include <bits/stdc++.h>
using namespace std;
const int maxN = 3e5 + 7;
const int maxK = 20;

string S; int N;

struct Suffix {
    int Pos;
    int Ft, Nt;

    bool operator < (const Suffix &X) {
        if (Ft != X.Ft) return Ft < X.Ft;
        return Nt < X.Nt;
    }
} L[maxN];

int P[2][maxN], SA[maxN], _2Pow[maxK];
```

```

int main()
{
    for (int i=0; i<20; ++i) _2Pow[i] = 1<<i;
    cin>>S; N = S.length();

    for (int i=0; i<N; ++i) P[0][i] = S[i] - 'a';
    int last = 0;
    for (int k=1; _2Pow[k]<N; ++k) {
        int now = 1 - last;
        for (int i=0; i<N; ++i) {
            L[i].Ft = P[last][i];
            L[i].Nt = i+_2Pow[k-1]<N? P[last][i+_2Pow[k-1]]: -1;
            L[i].Pos = i;
        }
        sort(L, L+N);
        for (int i=0; i<N; ++i) {
            P[now][L[i].Pos] = i>0 && L[i].Ft == L[i-1].Ft
                && L[i].Nt == L[i-1].Nt ? P[now][L[i-1].Pos] : i;
        }
        last = now;
    }
    for (int i=0; i<N; ++i) SA[i] = L[i].Pos;

    int ntest; cin>>ntest;
    while (ntest--) {
        string T; cin>>T; int nT = T.size();
        int ansL = N, l = 0, r = N-1;
        while (l<=r) {
            int m = (l+r)>>1;
            string X = S.substr(SA[m], nT);
            if (X < T) l = m + 1;
            else {
                if (X == T) ansL = m;
                r = m - 1;
            }
        }
        int ansR = -1; l = 0, r = N - 1;
        while (l<=r) {
            int m = (l+r)>>1;
            string X = S.substr(SA[m], nT);
            if (X > T) r = m - 1;
            else {
                if (X == T) ansR = m;
                l = m + 1;
            }
        }
        if (ansL <= ansR) cout<<ansR-ansL+1<<endl;
        else cout<<0<<endl;
    }
    return 0;
}

```

3.3. Đếm số lượng xâu con phân biệt độ dài K

Cho xâu S gồm n chữ cái in thường và số nguyên dương K . Đếm số lượng xâu con phân biệt có độ dài K trong xâu S ($1 \leq K \leq n \leq 10^5$).

Dữ liệu vào: Dòng đầu ghi số nguyên dương T là số lượng test, các dòng tiếp theo ghi thông tin các test. Thông tin mỗi test gồm hai dòng: dòng thứ nhất ghi hai số n, K , dòng thứ hai ghi xâu S . Tổng số chữ cái trong các test không quá 3×10^5 .

Kết quả: Mỗi truy vấn ghi trên một dòng là số lượng xâu con phân biệt độ dài K có trong S .

Ví dụ:

Input	Output
5	1
3 2	3
aaa	3
5 1	4
abcba	4
4 2	
abac	
10 2	
abbaaaabba	
7 3	
dogodog	

Link chấm: <https://www.spoj.com/problems/ADACLEAN/>

❖ *Hướng dẫn:*

Tạo LCP của xâu S . Khi đó số lượng xâu con phân biệt có độ dài K bằng $(n - K + 1)$ trừ cho số lượng $LCP[i] \geq K, i = 0..n - 1$.

❖ *Code mẫu:*

```
#include <bits/stdc++.h>
using namespace std;
const int maxN = 3e5 + 7;
const int maxK = 20;

int N, K; string S;
struct Suffix {
    int Ft, Nt, Pos;
    bool operator < (const Suffix &X) {
        if (Ft != X.Ft) return Ft < X.Ft;
        return Nt < X.Nt;
    }
} L[maxN];
```

```

int SA[maxN], LCP[maxN], _2Pow[maxK], P[maxK][maxN];
int Pos[maxN];

int main()
{
    for (int i=0; i<maxK; ++i) _2Pow[i] = 1 << i;

    int ntest; cin>>ntest;
    while (ntest--) {
        cin>>N>>K>>S;
        for (int i=0; i<N; ++i) P[0][i] = S[i] - 'a';

        for (int k=1; _2Pow[k-1]<N; ++k) {
            for (int i=0; i<N; ++i) {
                L[i].Ft = P[k-1][i];
                L[i].Nt = i+_2Pow[k-1]<N? P[k-1][i+_2Pow[k-1]] : -1;
                L[i].Pos = i;
            }
            sort(L,L+N);
            for (int i=0; i<N; ++i) {
                P[k][L[i].Pos] = i>0 && L[i].Ft==L[i-1].Ft &&
                    L[i].Nt==L[i-1].Nt? P[k][L[i-1].Pos] : i;
            }
        }

        for (int i=0; i<N; ++i) SA[i] = L[i].Pos;

        for (int i=0; i<N; ++i) Pos[SA[i]] = i, LCP[i] = 0;
        int k=0;
        for (int i=0; i<N; ++i) {
            if (Pos[i]==N-1) {
                k=0; continue;
            }
            int j = SA[Pos[i]+1];
            while (i+k < N && j+k < N && S[i+k]==S[j+k]) ++k;
            LCP[Pos[i]] = k;
            if (k) --k;
        }

        int ans = N-K+1;
        for (int i=0;i<N; ++i) ans -= (LCP[i]>=K);
        cout<<ans<<endl;
    }
    return 0;
}

```

3.4. Xâu con chung dài nhất

Cho hai xâu chỉ gồm các chữ cái in thường, độ dài mỗi xâu có không quá 25×10^4 . Tìm độ dài xâu con dài nhất xuất hiện đồng thời trong cả hai xâu.

Dữ liệu vào: Gồm hai dòng ghi lần lượt hai xâu.

Kết quả: Ghi độ dài xâu con dài nhất tìm được.

Ví dụ:

Input	Output
alsdfkjfj kdsal fdjskalajfkdsla	3

Link chấm: <https://www.spoj.com/problems/LCS/>

Hướng dẫn:

Ghép hai xâu lại với nhau tạo thành xâu S . Tạo LCP của xâu S . Khi đó độ dài xâu con chung dài nhất chính là giá trị của các $LCP[i]$ thoả mãn $SA[i] = SA[i+1]$ thuộc hai xâu khác nhau.

❖ Code mẫu:

```
#include <bits/stdc++.h>
const int maxN = 5e5 + 7;
const int maxK = 20;
using namespace std;

int P[maxK][maxN], SA[maxN], _2Pow[maxK], N;
struct Suffix {
    int Pos, Ft, Nt;
    bool operator < (const Suffix &X) const {
        if (Ft != X.Ft) return Ft < X.Ft;
        return Nt < X.Nt;
    }
    bool operator == (const Suffix &X) const {
        if (Ft != X.Ft) return false;
        return Nt == X.Nt;
    }
} L[maxN];

int LCP(int i, int j) {
    if (i==j) return N-i+1;
    int ans = 0;
    for (int k=maxK-1; k>=0; --k) {
        if (i+_2Pow[k]<=N && j+_2Pow[k]<=N&&P[k][i]==P[k][j]) {
            ans += _2Pow[k];
            i += _2Pow[k];
            j += _2Pow[k];
        }
    }
}
```

```

    }
}
return ans;
}

int main()
{
    ios::sync_with_stdio(false);
    for (int k=0; k<maxK; ++k) _2Pow[k] = 1 << k;

    string S,T; cin>>S>>T;
    int nS = S.size();
    S = S + "#" + T; N = S.size();

    for (int i=0; i<N; ++i) P[0][i] = S[i] - 'a';
    for (int k=1; _2Pow[k-1]<N; ++k) {
        for (int i=0; i<N; ++i) {
            L[i].Ft = P[k-1][i];
            L[i].Nt = i+_2Pow[k-1]<N?P[k-1][i+_2Pow[k-1]]:-1;
            L[i].Pos = i;
        }
        sort(L,L+N);
        for (int i=0; i<N; ++i) {
            P[k][L[i].Pos] = i>0 && L[i]==L[i-1] ?
                P[k][L[i-1].Pos]:i;
        }
    }
    for (int i=0; i<N; ++i) SA[i] = L[i].Pos;

    int ans = 0;
    for (int i=0; i<N-1; ++i) {
        int x = SA[i];
        int y = SA[i+1];
        if ( (x<nS && y>nS) || (x>nS && y<nS) )
            ans = max(ans, LCP(x,y));
    }
    cout<<ans;
    return 0;
}

```

3.5. Tìm xâu con nhỏ thứ K

Cho hai xâu S chỉ gồm các chữ cái in thường có độ dài không quá 10^5 . Hãy tìm xâu con nhỏ thứ K trong số các xâu con phân biệt của S .

Dữ liệu vào: Dòng đầu tiên ghi xâu S , dòng thứ hai ghi số nguyên dương Q là số lượng các yêu cầu cần tìm. Q dòng tiếp theo, mỗi dòng ghi một số nguyên dương K ($1 \leq K \leq 10^9, 1 \leq Q \leq 10^3$).

Kết quả: Ghi Q dòng lần lượt là kết quả của các truy vấn. Dữ liệu đảm bảo luôn tồn tại xâu kết quả.

Ví dụ:

Input	Output
abaab	aa
3	aba
2	baa
5	
10	

Link chấm: <https://www.spoj.com/problems/SUBLEX/>

Hướng dẫn:

Tạo mảng hậu tố SA và mảng tiền tố chung dài nhất của các hậu tố LCP của xâu S . Để thuận tiện cho bài toán này ta đặt $LCP[i]$ là tiền tố chung dài nhất của $SA[i]$ và $SA[i - 1]$. Khi đó số lượng xâu con phân biệt của hậu tố lớn thứ i theo thứ tự từ điển là $N - SA[i] - LCP[i]$. Các xâu con này cũng được sắp xếp tăng dần.

Gọi $C[i]$ là số lượng các xâu con phân biệt của i hậu tố đầu tiên. Khi đó muốn tìm xâu con thứ K ta chỉ cần tìm vị trí pos lớn nhất thoả mãn điều kiện $C[pos] \leq K$, lúc này xâu thứ K cần tìm là một tiền tố của hậu tố $SA[pos]$ có độ dài bằng $K - C[pos - 1] - LCP[pos]$.

❖ *Code mẫu:*

```
#include <bits/stdc++.h>
const int maxN = 1e5 + 7;
const int maxK = 20;
using namespace std;

int P[maxK][maxN], _2Pow[maxK], SA[maxN], LCP[maxN], N,
Pos[maxN], C[maxN];
string S;
struct Suffix{
    int Pos, Ft, Nt;

    bool operator <(const Suffix &X) const {
        if (Ft != X.Ft) return Ft < X.Ft;
        return Nt < X.Nt;
    }
    bool operator ==(const Suffix &X) const {
        if (Ft != X.Ft) return false;
        return Nt == X.Nt;
    }
} L[maxN];
```

```

int main()
{
    freopen("input.txt","r", stdin);
    ios_base::sync_with_stdio(false);
    for (int k=0; k<maxK; ++k) _2Pow[k] = 1<<k;

    cin>>S; N = S.length();

    for (int i=0; i<N; ++i) P[0][i] = S[i] - 'a';

    for (int k=1; _2Pow[k-1]<N; ++k) {
        for (int i=0; i<N; ++i) {
            L[i].Ft = P[k-1][i];
            L[i].Nt = i+_2Pow[k-1]<N ? P[k-1][i+_2Pow[k-1]]:-1;
            L[i].Pos = i;
        }
        sort(L,L+N);
        for (int i=0; i<N; ++i) {
            P[k][L[i].Pos] = i>0 && L[i]==L[i-1] ?
                P[k][L[i-1].Pos]:i;
        }
    }
    for (int i=0; i<N; ++i) SA[i] = L[i].Pos;

    for (int i=0; i<N; ++i) Pos[SA[i]] = i, LCP[i] = 0;
    int k=0;
    for (int i=0; i<N; ++i) {
        if (Pos[i]==N-1) {k=0; continue;}
        int j = SA[Pos[i]+1];
        while (i+k<N && j+k<N && S[i+k]==S[j+k]) ++k;
        LCP[Pos[j]] = k;
        if (k) --k;
    }

    for (int i=0; i<N; ++i) C[i] = N - SA[i] - LCP[i];
    for (int i=1; i<N; ++i) C[i] += C[i-1];

    int ntest; cin>>ntest;
    while(ntest--) {
        int p; cin>>p;
        int pos = lower_bound(C,C+N,p) - C;
        int len = p + LCP[pos] - C[pos-1];
        cout<<S.substr(SA[pos],len)<<endl;
    }

    return 0;
}

```

C. KẾT LUẬN

Mảng hậu tố là một cách tiếp cận để giải quyết lớp các bài toán xử lí xâu khá hiệu quả trong các kỳ thi học sinh giỏi. Việc cài đặt mảng hậu tố khá đơn giản và dễ nhớ.

Những bài toán được lựa chọn trình bày trong chuyên đề này mới chỉ dùng lại ở mức áp dụng trực tiếp các tính chất của mảng hậu tố chứ chưa phải là những bài toán phức tạp, cần sử dụng kết hợp thêm các kiểu dữ liệu nâng cao khác như cây phân đoạn, cây nhị phân tìm kiếm... để giải như bài “dãy ngoặc đúng” (<http://vnoi.info/problems/DBRACKET/>) hoặc bài “đếm số lượng xâu con xuất hiện ít nhất K lần” (<https://www.codechef.com/problems/ANUSAR>)... Các bài toán trong chuyên đề này có mục đích là để kiểm chứng tính đúng đắn và hiệu quả thuật toán nên chỉ dùng lại ở cách chấm online chứ chưa chuẩn bị được đầy đủ các bộ test để quý thầy cô sử dụng được ngay, rất mong quý thầy cô thông cảm.

Do thời gian và năng lực còn nhiều hạn chế nên chuyên đề này không thể tránh khỏi những thiếu sót. Kính mong nhận được những góp ý, chỉ bảo của quý thầy cô!

D. TÀI LIỆU THAM KHẢO

- <http://vnoi.info/wiki/algo/string/hash>
- <http://vnoi.info/wiki/algo/data-structures/suffix-array>
- <https://cp-algorithms.com/string/suffix-array.html>
- <https://codeforces.com/blog/entry/15729>
- <https://www.hackerearth.com>
- <https://www.codechef.com>

Người viết chuyên đề: Hoàng Văn Diệu (Số ĐT: 0982081481)

– THPT Chuyên Lê Quý Đôn – Quảng Trị