3. SUFFIX ARRAY

3.1. Một số khái niệm cơ sở

Gọi Σ là một tập hợp hữu hạn có thứ tự được gọi là bảng chữ cái (*alphabet*), các phần tử ∈ Σ được gọi là các ký tự. Σ^* là tập các xâu (*string*) gồm các ký tự ∈ Σ. Có thể coi mỗi xâu ∈ Σ^* là một dãy hữu hạn các ký tự ∈ Σ. Ký hiệu ϵ là xâu rỗng, tập các xâu khác rỗng được gọi là $\Sigma^+ = \Sigma^* - \{\epsilon\}$.
Chiều dài của một xâu x, ký hiệu $|x|$ là số ký tự trong xâu x. Các ký tự trong xâu x được đánh số từ 1 đến $|x|$: $x = x_1x_2 \dots x_{|x|}$.
Xâu nối của hai xâu x, y ký hiệu xy có chiều dài $|x| + |y|$ và tạo thành bằng cách lấy các ký tự trọng x sau đó nối tiếp bằng các ký tự trong y.
Ta gọi xâu w là tiền tố (*prefix*)của x, ký hiệu $w \sqsubset x$, nếu tồn tại xâu y để $x = wy$. Xâu w được gọi là hậu tố (*suffix*) cua xâu x, ký hiệu $w \sqsupset x$ nếu tồn tại xâu y để $x = yw$. Dễ thấy một xâu w là tiền tố hoặc hậu tố của x thì $|w| \leq |x|$. Một xâu có thể vừa là tiền tố vừa là hậu tố của một xâu khác.
Hai quan hệ $\sqsubset, \sqsupset$ có tính chất bắc cầu, có nghĩa là:

- Nếu $x \sqsubset y$, và $y \sqsubset z$ thì $x \sqsubset z$
- Nếu $x \sqsupset y$, và $y \sqsupset z$ thì $x \sqsupset z$.

Bổ đề: (Về tính gối nhau của các tiền tố và hậu tố

- Nếu x, y cùng là tiền tố của z thì x sẽ là tiền tố của y nếu $|x| \leq |y|$, y sẽ là tiền tố của x nếu $|y| \leq |x|$
- Nếu x, y cùng là hậu tố của x thì x là hậu tố của y nếu $|x| \leq |y|$, y là hậu tố của x nếu $|y| \leq |x|$.

Cho $T = t_1t_2 \dots t_n$ và $P = p_1p_2 \dots p_m$ là hai xâu ký tự. Ta nói xâu P xuất hiện trong xâu T tại vị trí k nếu $P = t_kt_{k+1} \dots t_{k+m-1}$. Nếu xâu P xuất hiện trong xâu T tại một vị trí nào đó thì P là xâu con (***substring***) của T. Một cách định nghĩa khác về xâu con của T: P là một tiền tố của một hậu tố của T
Trong chuyên đề này qui ước $\Sigma = \{@, A..Z\}$

2. Mảng hậu tố

Cho một xâu $T = t_1t_2 \dots t_n \in \Sigma^+$, qui ước có duy nhất $t_n = @$. Mảng hậu tố (*suffix array*) của T, ký hiệu là $SA(T)$ là thứ tự từ điển của tất cả các hậu tố của T.
Một hậu tố $t_{i\dots n}$ có thể đồng nhất với vị trí i. Khi đó mảng hậu tố của T có thể biểu diễn như là một hoán vị $(sa_1, sa_2, \dots, sa_n)$ của $(1,2, \dots, n)$ sao cho:

$$t_{sa_1\dots n} < t_{sa_2\dots n} < \dots < t_{sa_n\dots n}$$

(quan hệ '<' là quan hệ theo thứ tự từ điển)
Ví dụ với $T = BANANA@$. Ta có:

1	BANANA@
2	ANANA@
3	NANA@
4	ANA@
5	NA@
6	A@
7	@

7	@
6	A@
4	ANA@
2	ANANA@
1	BANANA@
5	NA@
3	NANA@

Để xây dựng mảng hậu tố ta có thể tiến hành sắp xếp trực tiếp trên các hậu tố. Tuy nhiên việc này chỉ áp dụng khi độ dài của xâu nhỏ bởi vì phép so sánh hai xâu đã có độ phức tạp $O(|T|)$. Ta sẽ khảo sát một phương pháp tốt hơn: **Thuật toán nhân đôi tiền tố**:

Khởi tạo: Sắp xếp các hậu tố của T theo thứ tự tăng dần của ký tự đầu tiên. Điều này tương đương với việc sắp xếp các ký tự của T tăng dần. Sau đó ta gán cho mỗi hậu tố một số nguyên ∈ $[1, n]$ thỏa

mãn: hai hậu tố có ký tự đầu bằng nhau thì mang khóa bằng nhau, hai hậu tố có ký tự đầu khác nhau phải mang khóa khác nhau và hậu tố nào có ký tự đầu nhỏ hơn phải mang khóa nhỏ hơn. Việc gán khóa số mất thời gian $O(n)$. Khóa số là đại diện cho các ký tự đầu của các hậu tố, tức là dãy các hậu tố xếp theo thứ tự tăng dần của khóa số cũng chính là dãy hậu tố theo thứ tự tăng dần của ký tự đầu tiên.

Phần chính của thuật toán được thực hiện qua nhiều pha, tại mỗi pha, giả thiết là đã có dãy các hậu tố xếp theo thứ tự tăng dần của k ký tự đầu cùng các khóa số tương ứng với thứ tự sắp xếp, thuật toán sẽ xây dựng dãy hậu tố xếp theo thứ tự tăng dần của $2k$ ký tự đầu và dãy khóa số tương ứng.

- Gọi các khóa đang được gán cho các hậu tố là các khóa sơ cấp (*primary keys*), mỗi hậu tố sẽ được bổ sung một khóa nữa được gọi là khóa thứ cấp (*secondary keys*). Khóa thứ cấp của hậu tố $t_{i\ldots n}$ là khóa sơ cấp của hậu tố $t_{i+k,\ldots,n}$ hoặc bằng 0 nếu $i+k>n$
- Sắp xếp lại các hậu tố theo qui tắc: Trước tiên sắp xếp tăng dần các khóa sơ cấp, nếu hậu tố có hai khóa sơ cấp bằng nhau thì hậu tố có khóa thứ cấp nhỏ hơn sẽ được xếp trước. Theo giả thiết về dãy khóa sơ cấp và cách xây dựng khóa thứ cấp ta sẽ được dãy các hậu tố xếp theo thứ tự tăng dần của $2k$ ký tự đầu sau khi xếp (khi hai hậu tố có k ký tự đầu khớp nhau thì k ký tự sau quyết định hậu tố nào đứng trước)
- Với các hậu tố đã sắp xếp, mỗi hậu tố được gán khóa số mới: Hậu tố đứng đầu được đánh số 1. Bắt đầu từ hậu tố thứ hai trở đi trong dãy, nếu nó có cả khóa sơ cấp và thứ cấp giống với hậu tố liền trước nó thì khóa số mới của nó bằng khóa số mới của hậu tố liền trước, nếu không thì khóa số mới của nó bằng khóa số mới của hậu tố liền trước cộng thêm 1. Thao tác gán số mới mất $O(n)$.

Như vậy các bước lặp lần lượt xây dựng được thứ tự các hậu tố xếp theo 1, 2, 4, 8, 16, 32, ... ký tự đầu tiên. Mảng hậu tố sẽ thu được sau bước lặp thứ $\lceil \log_2 n \rceil$. Thuật toán có thể kết thúc sớm tại một bước nào đó khi tất cả các khóa gán cho hậu tố là các số nguyên phân biệt từ 1 đến n.

Đoạn code dưới đây sử dụng lệnh sort của C++ để sắp xếp (độ phức tạp $O(nlog^2n)$):

```
using namespace std;
typedef pair<int,int> II;
typedef pair<II,int> III;

int n;                          // độ dài xâu
char T[maxn];                   // Xâu T[1]...T[n]
int sa[maxn];                   // Mảng hậu tố

// Thủ tục khởi tạo sa[1..n] khi k=1
III p[maxn];                    // Mảng cặp ((khóa sơ cấp,khóa thứ cấp), chỉ số)
bool mark[maxn];                // Đánh dấu khác với hậu tố liền trước
int key[maxn];                  // Mảng giá trị khóa
void InitSuff() {
    for(int i=1;i<=n;i++) p[i]=make_pair(make_pair(T[i],0),i);
    sort(p+1,p+n+1);
    for(int i=1;i<=n;i++) sa[i]=p[i].SC;
    mark[1]=true;
    for(int i=2;i<=n;i++) mark[i]=(p[i].FT!=p[i-1].FT);
}

// Thủ tục chính xây dựng sa[1...n]
void SuffArr() {
    for(int k=1;k<n;k*=2) {
        // Tao mang gia tri
        int val=0;
        for(int i=1;i<=n;i++) {
```

```
            if (mark[i]) val++;
            key[sa[i]]=val;
        }
        if (val==n) break;
        // Lap mang so cap - thu cap moi{
        for(int i=1;i<=n;i++) {
            int t=(i+k<=n) ? key[i+k]: 0;
            p[i]=make_pair(make_pair(key[i],t),i);
        }
        sort(p+1,p+n+1);
        for(int i=1;i<=n;i++) sa[i]=p[i].SC;
        // Danh dau khac nhau
        mark[1]=true;
        for(int i=2;i<=n;i++) mark[i]=(p[i].FT!=p[i-1].FT);
    }
}
```

Sử dụng sắp xếp đếm phân phối với độ phức tạp $O(n)$ ta có thuật toán $O(n \log n)$:

```
// Suffix array sử dụng đếm phân phối O(nlogn)
// Input:
char s[maxn];       // Xâu ký tự tìm suffix. Chú ý kết thúc s[i]=0 -ký tự cầm canh
int n;              // Độ dài của xâu, tính cả ký tự cầm canh
// Output:
int sa[maxn];       // Mảng suffix array

// Các mảng phụ trợ
int cnt[256];       // Mảng đếm phân phối (ký tự ASCCI mã từ 0…255 (0 là cầm canh)
bool mark[maxn];    // mark[i]=true ↔ tiền tố của sa[i] khác với tiền tố sa[i-1]
int key[maxn];      // Mảng qui đổi giá trị tiền tố thuộc [1…n]
int head[maxn];     // Mảng đếm phân phối sử dụng khi độ dài tiền tố >1
int sb[maxn];       // Mảng lưu khóa thứ cấp
int rk[maxn];       // Mảng tính hạng của tiền tố thứ i: rk[i]=j ↔ sa[j]=i

// Thủ tục khởi đầu  suffix array (k=1)
void InitSuffix() {
    memset(cnt,0,sizeof(cnt));
    for(int i=1;i<=n;i++) cnt[int(s[i-1])]++;
    for(int i=0;i<256;i++) cnt[i]+=cnt[i-1];
    for(int i=n;i>=1;i--) {
        int u=s[i-1];
        sa[cnt[u]]=i; cnt[u]--;
    }
    mark[1]=true;
    for(int i=2;i<=n;i++) mark[i]=(s[sa[i]-1]!=s[sa[i-1]-1]);
}

// Thủ tục nhân đôi tiền tố
void SuffixArray() {
    for(int k=1;k<n;k*=2) {
        // Đầu tiên tương ứng mỗi giá trị sa[i] với key[sa[i]] trong đoạn [1..n]
        // Với mỗi sa[i], sb[i] là k ký tự phía trước nó. Nếu chuyển sang độ dài 2k thì
        // (sb[i],sa[i]) là một cặp tiền tố độ dài 2k. Chú ý do tính ổn định của sắp
       // xếp phân phối và do sa[i] đã được sắp nên việc sắp sb[i] kéo theo sắp cả cặp
        int val=0;
        for(int i=1;i<=n;i++) {
            if (mark[i]) val++;
```

```
            key[sa[i]]=val;
            sb[i]=sa[i]-k;
            if (sb[i]<1) sb[i]+=n;
        }
        if (val==n) break;
        // Sắp lại sb[i], gán cho sa[i] bằng đếm phân phối
        memset(head,0,sizeof(head));
        for(int i=1;i<=n;i++) head[key[sb[i]]]++;
        for(int v=2;v<=val;v++) head[v]+=head[v-1];
        for(int i=n;i>=1;i--) {
            int v=key[sb[i]];
            sa[head[v]]=sb[i]; head[v]--;
        }
        // Tinh lai mang mark
        val=0;
        for(int i=1;i<=n;i++) {
            int j=sa[i]+k;
            if (j>n) j-=n;
            if (key[j]!=val) {
                mark[i]=true;
                val=key[j];
            }
        }
    }
}
```

3.3.Mảng tiền tố chung dài nhất

Tiền tố chung dài nhất (*longgest common prefix*) của hai xâu x, y là xâu z có độ dài lớn nhất thỏa mãn z vừa là tiền tố của x, vừa là tiền tố của y.

Cho $T = t_1 t_2 \dots t_n$ là một xâu khác rỗng, $SA(T) = (sa_1, sa_2, \dots, sa_n)$ là mảng hậu tố của T. Mảng tiền tố chung dài nhất $LCP(T)$ là dãy số nguyên $(lcp_1, lcp_2, \dots, lcp_n)$ thỏa mãn:

- $lcp_1 = 0$
- $\forall i > 1$: lcp_i là độ dài tiền tố chung dài nhất giữa hậu tố $t_{sa_i \dots n}$ và $t_{sa_{i-1} \dots n}$

Ví dụ với $T = BANANA@$, mảng hậu tố của T là (7,6,4,2,1,5,3) ta có:

$$lcp_1 = 0, lcp_2 = 0, lcp_3 = 1, lcp_4 = 3, lcp_5 = 0, lcp_6 0, lcp_7 = 2$$

Bài toán: Cho xâu $T = t_1 t_2 \dots t_n \in \Sigma^+$ trong đó có duy nhất $t_n = @$ cùng với mảng hậu tố tương ứng $SA(T) = (sa_1, sa_2, \dots, sa_n)$. Hãy xây dựng mảng tiền tố chung dài nhất

$$LCP(T) = (lcp_1, lcp_2, \dots, lcp_n)$$

Thuật toán phổ biến nhất là thuật toán Kasai:

Trước tiên chúng ta tính mảng $rank_1, \dots, rank_n$ với $rank_i = j \iff sa_j = i$

Mảng lcp được xây dựng theo thứ tự:

$$lcp[rank_1], lcp[rank_2], \dots, lcp[rank_n]$$

Với mỗi giá trị i, gọi $q = lcp[rank_i]$ và $j = sa[rank_i - 1]$ là hậu tố đứng liền trước hậu tố i trong mảng hậu tố. Theo định nghĩa về mảng $LCP(T)$ ta có q là độ dài tiền tố chung dài nhất giữa $t_{i,\dots,n}$ và $t_{j,\dots,n}$. Loại bỏ ký tự đầu tiên của hai hậu tố này ta có tiền tố chung giữa hai hậu tố $t_{i+1,\dots,n}$ và $t_{j+1,\dots,n}$ có độ dài $q - 1$ nếu $q \geq 1$ và bằng 0 trong trường hợp ngược lại.

Vì hậu tố j đứng liền trước hậu tố i trong mảng hậu tố nên $t_{j,\dots,n} < t_{i,\dots,n}$. Nếu $q \geq 1$ thì $t_j = t_i$ nên nếu loại bỏ ký tự đầu giống nhau từ hai hậu tố này thì $t_{j+1,\dots,n}$ vẫn nhỏ hơn $t_{i+1,\dots,n}$. Xét trên thứ tự từ điển của mảng hậu tố $t_{j+1,\dots,n}$ có thể không đứng liền trước $t_{i+1,\dots,n}$ nhưng bởi tiền tố chung dài nhất của chúng có độ dài $q - 1$ nên mọi hậu tố nằm giữa chúng trong mảng hậu tố đều phải có $q - 1$ ký tự đầu trùng với $t_{j+1,\dots,n}$ cũng như $t_{i+1,\dots,n}$. Điều này chỉ ra rằng $t_{i+1,\dots,n}$ có ít nhất $q - 1$ ký tự đầu

tiên trùng với hậu tố đứng liền trước nó trong mảng hậu tố. Hay $lcp[rank_{i+1}] \geq q - 1$. Dĩ nhiên bất đẳng thức trên đúng ngay cả khi $q = 0$. Từ đó ta có bổ đề sau:

Bổ đề 2: Với $\forall\, i: 1 \leq i < n$ ta có $lcp[rank_{i+1}] \geq lcp[rank_i] - 1$

Bổ đề trên cho ta thủ tục tìm mảng lcp như sau:

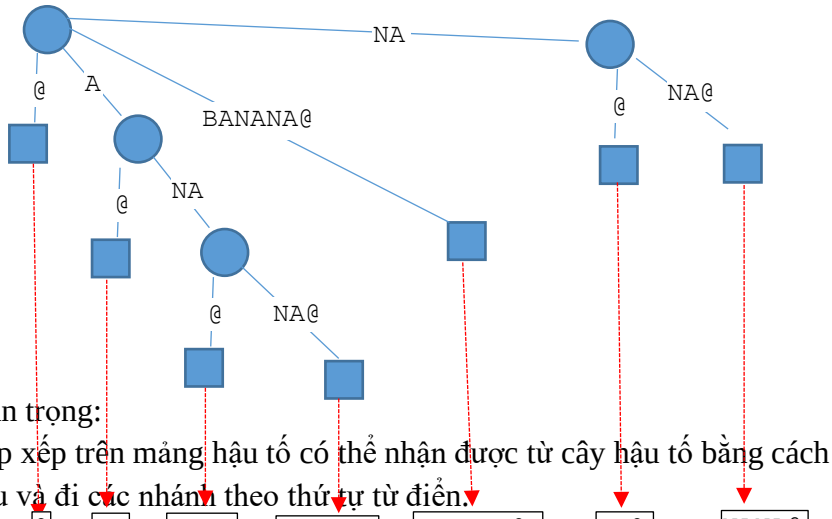
```
// Thủ tục tính mảng tiền tố chung dài nhất
int rk[maxn];
void LcpArray() {
    for(int i=1;i<=n;i++) rk[sa[i]]=i;
    lcp[1]=0;
    int q=0;
    for(int i=1;i<=n;i++) if (rk[i]>1) {
        int j=sa[rk[i]-1];
        while (s[i+q]==s[j+q]) q++;
        lcp[rk[i]]=q;
        q--;
        if (q<0) q=0;
    }
}
```

3.4. Cây hậu tố (Suffix Tree)

Cây hậu tố của một xâu $T \in \Sigma^+$, ký hiệu $ST(T)$ là một cấu trúc dữ liệu dạng cây có tính chất sau:

- Mỗi cạnh của cây có nhãn là một xâu $\in \Sigma^+$. Các cạnh đi từ một nút xuống các nút con của nó phải mang nhãn là các xâu có ký tự đầu tiên hoàn toàn phân biệt.
- Mỗi nút v của cây hậu tố cũng mang một nhãn, nhãn của nút v, kỹ hiệu $\bar{v}$ là xâu tạo thành bằng cách nối tiếp các nhãn cạnh trên đường đi từ gốc xuống nút v. $|\bar{v}|$ được gọi là độ sâu của nút v, ký hiệu $depth(v)$.
- Mỗi hậu tố của T là nhãn của một nút lá

Ví dụ: $T = BANANA@$ ta có cây hậu tố như sau:



Một vài nhận xét quan trọng:

1. Các hậu tố sắp xếp trên mảng hậu tố có thể nhận được từ cây hậu tố bằng cách duyệt cây theo chiều sâu và đi các nhánh theo thứ tự từ điển.
2. lcp_i là độ sâu của chế chung gần nhất hai lá chứa hậu tố suf_{i-1} và suf_i
3. Nhãn của một nút bất kỳ luôn là tiền tố chung dài nhất của tất cả các hậu tố có lá nằm trên cây con nút này

Một đặc điểm thú vị khi giải các bài toán sử dụng $suffix\ array$ là ta **luôn tư duy trên cây hậu tố và triển khai trên mảng SA(T), LCP(T) các truy vấn mảng**

3.5. Một số ứng dụng điển hình

3.5.1. Xâu con lặp dài nhất

Cho xâu ký tự T. Hãy tìm xâu P dài nhất sao cho P xuất hiện trong T ít nhất k lần

Thuật toán:

Một xâu con có thể xem như là tiền tố của một hậu tố. Dựng cây hậu tố của T thì một xâu con là một nhãn của một nút nào đó trong cây này. Điều kiện xâu con xuất hiện ít nhất k lần tương tương với điều kiện số nút lá trong cây con gốc nhãn này $\geq k$. Nếu xâu con có độ dài q thì trên mảng tiền tố chung dài nhất phải tồn tại một đoạn $k-1$ phần tử liên tiếp có giá trị $\geq q$. Do đó ta có thuật toán:
B1: Lập mảng tiền tố chung dài nhất $LCP(T) = (lcp_1, lcp_2, \dots, lcp_n)$
B2: Với mỗi $i: 1 \leq i \leq n-(k-1)+1$ tìm $q_i = \min(lcp_i, \dots, lcp_{i+k-2})$. Có thể sử dụng Dequeue để thực hiện điều này. Đáp số là $\max(q_1, \dots, q_{n-k+2})$.

3.5.2. Số xâu con phân biệt

Cho xâu ký tự S, nếu ta đem một dãy các ký tự liên tiếp của S nối lại thì được một xâu con của S. Hãy cho biết S có bao nhiêu xâu con khác rỗng phân biệt?
Ví dụ với $S = AAABB$ ta có 11 xâu con:
A, B, AA, AB, BB, AAA, AAB, ABB, AAAB, AABB, AAABB

Thuật toán:

Bài toán đơn thuần là chỉ đếm số nút trên cây trie hậu tố của S ngoại trừ nút gốc (vì mỗi xâu con phân biệt sẽ tương ứng với một nút của trie hậu tố). Tuy nhiên ở đây chúng ta không cài đặt chương trình theo cách trên mà sử dụng mảng hậu tố (suffix array):
Giả sử ra có mảng hậu tố $(sa_1, sa_2, \dots., sa_n)$ và mảng tiền tố chung dài nhất $(lcp_1, lcp_2, \dots, lcp_n)$. Phân tích quá trình hậu tố $s_{sa_1 \dots n}$ được chèn vào trie thì lcp_i ký tự đầu tiên được chèn không phát sinh ra nút và cạnh. Những ký tự sau, mỗi ký tự sẽ thêm một cạnh và một nút trên trie. Suy ra đáp số là:

$$\sum_{i=1}^{n}(n - sa_i + 1 - lcp_i) = n(n+1) - \sum_{i=1}^{n} sa_i - \sum_{n=1}^{n} lcp_i = \frac{n(n+1)}{2} - \sum_{i=1}^{n} lcp_i$$

Vì $\sum_{i=1}^{n} sa_i = \frac{n(n+1)}{2}$.

3.5.3. Xâu ngắn nhất

Cho hai xâu ký tự A và B chỉ gồm các chữ cái tiếng Anh in hoa. B không phải là một xâu ký tự con của A. Hãy tìm một xâu ký tự C ngắn nhất thỏa mãn:

- C là xâu ký tự con của B
- C không là xâu ký tự con của A

Thuật toán:

Lập xâu $C = AB = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$. Khi đó mọi xâu con của B đều là một hậu tố của C có chỉ số $> n$. Lập mảng hậu tố của C: $SA(C) = (sa_1, sa_2, \dots, sa_{n+m})$ và mảng tiền tố chung dài nhất $LCP(C) = (lcp_1, lcp_2, \dots, lcp_{n+m})$.
Xét trên cây Trie hậu tố của C. Một xâu con của B sẽ tương ứng với một nút trên cây này. Xâu con này sẽ không là xâu con của A nếu như cây con với gốc này không chứa một hậu tố $sa_i \leq n$. Độ sâu của nút tương ứng với chiều dài của xâu con. Tất nhiên độ sâu giảm khi số nút lá càng nhiều. Do vậy chúng ta chỉ xét các đoạn liên tiếp dài nhất chỉ chứa các hậu tố $> n$ và lấy min của các lcp trên đoạn này. Giá trị nhỏ nhất chính là độ dài của xâu cần tìm

3.5.4 Xâu con chung

Cho n xâu ký tự $T_1, T_2, \dots, T_n$. Hãy tìm xâu con chung khác rỗng dài nhất của n xâu trên (xâu con xuất hiện trong cả n xâu)

Thuật toán:

Đặt $S = T_1 + T_2 + \dots + T_n$. Gọi $SA(S), LCP(S)$ lần lượt là mảng hậu tố và mảng tiền tố chung dài nhất. Chú ý rằng mỗi hậu tố, tùy theo giá trị chỉ số của nó sẽ thuộc vào một trong các loại từ 1 đến n (loại i tương ứng với hậu tố của T_i). Một xâu con chung của n xâu tương ứng với một nút trên $ST(S)$ trong đó có n lá thuộc về n loại hậu tố. Bài toán qui về bài toán với mỗi $i: 1 \leq i \leq |S|$. Tìm dãy con

ngắn nhất $lcp_i, \dots, lcp_k$ có đủ n loại hậu tố. Truy vấn min $(lcp_i, \dots, lcp_k)$ và lấy giá trị lớn nhất trong các truy vấn.

3.5.5. Xâu đối xứng dài nhất

Cho một xâu ký tự T chỉ gồm các chữ cái tiếng Anh in thường. Hãy tìm xâu đối xứng dài nhất gồm các ký tự liên tiếp của T.

Thuật toán:

Giả sử $T = t_1 t_2 \dots t_n$. Đặt $\bar{T} = t_n t_{n-1} \dots t_2 t_1$ (xâu nhận được từ T bằng cách viết các ký tự theo chiều Lập xâu ghép $S = T + \bar{T} + '@'$. Xây dựng mảng hậu tố $SA(S)$, mảng tiền tố chung dài nhất $LCP(S)$. Xét hai trường hợp:

TH1: Xâu đối xứng có độ dài lẻ. Khi đó nó luôn được viết dưới dạng $\bar{A} t_i A$ trong đó A là một xâu con của T bắt đầu từ vị trí $i+1$ đồng thời cũng là xâu con của $\bar{T}$ bắt đầu từ vị trí $n-i$. Bài toán qui về tìm tiền tố chung dài nhất giữa $rank_{i+1}$ và $rank_{2n-i}$. Có thể thực hiện truy vấn này trong $O(1)$ nếu trước đó dựng bảng RMQ trên $LCP(S)$

TH2: Xâu đối xứng độ dài chẵn. Khi đó nó luông có dạng $\bar{A}A$ trong đó A là xâu con của T bắt đầu từ vị trí i còn đồng thời cũng là xâu con của $\bar{T}$ bắt đầu từ vị trí $n-i$ và bài toán cũng được qui dẫn tương tự như trên.

3.5.6. Mẫu ghép

Cho xâu ký tự A độ dài $n \le 10^5$, tìm xâu B ngắn nhất sao cho mọi ký tự trong A đều tồn tại một xâu con nào đó đúng bằng B chứa vị trí ký tự đó. Hay nói cách khác A là một ghép nối của một loạt xâu B (các xâu ghép nối có thể phủ chồng lên nhau ở một số đoạn)

Ví dụL

```
ababbababbababbabaaababbaba (A)
ababbaba (B)
     ababbaba
          ababbaba
                   ababbaba
```

Thuật toán:

Dễ nhận thấy rằng B phải là một tiền tố của A. Giả sử rằng đã tìm được xâu B, Đánh dấu các vị trí bắt đầu của B trong sơ đồ ghép bằng số 1, các vị trí còn lại bằng số 0. Chú ý rằng khi đó mỗi vị trí số 0 đều được phủ bởi ít nhất một bản sao của B. Do vậy dãy số 0 liên tiếp không được vượt quá chiều dài của dãy B. Ngược lại nếu với mỗi tiền tố B của A ta đánh dấu vị trí xuất hiện của nó trong A bởi số 1 thì nếu dãy số 0 liên tiếp không vượt quá độ dài của B thì các vị trí 0 đều được phủ bởi một bản sao của B. Do đó ta có thuật toán:

Xây dựng mảng hậu tố $SA(A) = (sa_1, sa_2, \dots, sa_n)$. Khởi tạo dãy B rỗng, lần lượt thêm vào dãy B các ký tự $a_1, a_2, \dots$ với mỗi lần như vậy:

- Tìm khoảng L, R sao cho $sa_L, sa_{L+1}, \dots, sa_R$ đều có $|B|$ ký tự đầu giống B. Việc tìm L, R có thể thực hiện bằng tìm kiếm nhị phân. Chú ý rằng các khoảng L, R của lần lặp sau nằm trong khoảng L, R của lần lặp trước.
- Mỗi lần, các hậu tố ở ngoài khoảng $[L, R]$ được gán 0 và các hậu tố trong khoảng này được gán 1. Tìm dãy số 0 liên tiếp dài nhất. Nếu dãy này có độ dài $< |B|$ thì dừng thuật toán

Để truy vấn, có thể sử dụng segment tree quản lý dãy số 0 liên tiếp.

5.9. Mật mã ẩn (HIDECODE.*)

Cho xâu ký tự S . Tìm hoán vị vòng quanh của S có thứ tự từ điển nhỏ nhất

Input: Một dòng chứa xâu S chỉ gồm các chữ cái tiếng Anh in hoa có độ dài không quá 10^5

Output: Hoán vị vòng quanh nhỏ nhất tìm được

Example:

Input	Output
ALABALA	AALABAL

Thuật toán:

Tạo xâu ghép $T = S + S$. Lập mảng hậu tố của T. Tìm sa_i đầu tiên thỏa mãn $1 \le sa_i \le n$ với n là độ dài của xâu.

10. String Reconstruction (STRREC.*)

http://vn.spoj.com/problems/C11SSTR/

Cho một xâu ký tự $S = s_1 s_2 \ldots . s_n$ chỉ gồm các chữ cái tiếng Anh in thường. Lập các xâu hậu tố

$T_1 = s_1 s_2 \ldots s_n$

$T_2 = s_2 \ldots s_n$

...

$T_i = s_i \ldots s_n$

...

$T_n = s_n$

Sau đó ta sắp xếp các xâu này tăng dần:

$$T_{sa_1} \le T_{sa_2} \le \cdots T_{sa_n}$$

Mảng $SA = (sa_1, sa_2, \ldots, sa_n)$ được gọi là mảng hậu tố (suffix array) của xâu S.

Yêu cầu: Cho trước mảng SA và số nguyên dương K. Trong số các xâu S có suffix array là mảng SA hãy tìm xâu có thứ tự từ điển thứ K.

Input:

- Dòng đầu tiên ghi hai số nguyên dương n, K - độ dài của xâu và số K
- Dòng thứ hai ghi n số nguyên dương phân biệt $sa_1, sa_2, \ldots, sa_n$

Output: Xâu ký tự tìm được ghi trên một dòng (chỉ gồm các chữ cái tiếng Anh in thường). Nếu không tồn tại xâu như vậy thì ghi -1

Example:

Input	Output
4 2 3 1 4 2	bdab

Giải thích: 3 xâu đầu tiên theo thứ tự từ điển nhận suffix array đã cho là:

bcab

bdab

beab

Do $K = 2$ nên xâu cần tìm là 'bdab'

Subtasks:

- Subtask 1: $n \le 20, K = 1$ [20%]
- Subtask 2: $n \le 20, K \le 1000$ [10%]
- Subtask 3: $n \le 20, K \le 10^{12}$ [10%]
- Subtask 4: $n \le 1000, K = 1$ [20%]
- Subtask 5: $n \le 1000, K \le 1000$ [20%]
- Subtask 6: $n \le 1000, K \le 10^{12}$ [20%]