MỤC LỤC

ỨNG DỤNG CỦA CTDL PALINDORME TREE TRONG LỚP CÁC BÀI TOÁN VỀ XÂU PALINDROME

Nguyễn Thị Vân Khánh
THPT Chuyên Biên Hòa – Hà Nam
(Chuyên đề đạt giải nhì)

I. LỜI NÓI ĐẦU

Trong Toán học, khi giải một bài toán, người học chỉ cần đưa ra một cách giải đúng để cho đáp số đúng là lời giải được chấp nhận. Nhưng trong Tin học thì có một điểm khác biệt hơn trong Toán học. Đó là, khi lập trình để giải một bài toán thì vấn đề đặt ra cho mỗi người lập trình không phải chỉ là cho ra đáp số đúng, mà vấn đề quan trọng hơn là phải đưa ra đáp số đúng trong thời gian ngắn nhất (mà điều này được thể hiện thông qua độ phức tạp thuật toán). Thông thường, để đạt được độ phức tạp thuật toán như mong muốn, người lập trình sẽ tìm ra một thuật toán ban đầu làm cơ sở, rồi từ đó dùng các kỹ năng để giảm độ phức tạp của thuật toán. Xâu Palindrome là một dạng bài toán kinh điển trong Tin học. Để giải quyết các bài toán liên quan đến xâu Palindrome đã có rất nhiều thuật toán được đưa ra. Cũng có nhiều thuật toán hay, cải tiến nhằm tối ưu thuật toán. Nhưng đối với bài toán xâu Palindrome có kích thước lớn thì vẫn đòi hỏi mất một thời gian chạy là $O(n)$ (chẳng hạn thuật toán Manacher). Đối với nhiều bài toán về xâu Palindrome nhưng với kích thước dữ liệu lớn, nếu cài đặt bằng các kỹ thuật thông thường thì thuật toán sẽ chạy mất nhiều thời gian. Nhưng nếu ta ứng dụng cấu trúc dữ liệu *Palindrome Tree (cây Palindrome)* vào để giải quyết thì độ phức tạp bài toán chỉ là tuyến tính $O(n)$. Đây là một cấu trúc dữ liệu khá hay và mới, thường được áp dụng vài giải các bài toán lập trình trong các kỳ thi chọn học sinh giỏi quốc gia và quốc tế.

Hiện nay, các bài toán về xâu Palindrome khá nhiều, ứng dụng giải các bài toán về xâu Palindrome mà có sử dụng cấu trúc dữ liệu cây Palindrome trong các kì thi học sinh giỏi môn tin học cũng được áp dụng nhiều, nhưng tài liệu viết một cách chi tiết, hệ thống về cấu trúc dữ liệu này thì chưa có. Điều này làm cho việc nghiên cứu, giảng dạy và học về cấu trúc dữ liệu này khá khó khăn. Là một giáo viên với nhiều năm kinh nghiệm dạy các đội tuyển học sinh giỏi thi tỉnh và quốc gia, tôi thấy rằng các bài toán về xâu Palindrome với dữ liệu nhỏ học sinh giải quyết khá nhuần nhuyễn bằng thuật toán DP với độ phức tạp là $O(n^2)$ (với n là độ dài của xâu đã cho) hoặc tối ưu hơn học sinh có thể giải quyết bài toán bằng thuật toán phức tạp Manacher với độ phức tạp tuyến tính $O(n)$. Việc ứng dụng thuật toán Manacher không dễ dàng với nhiều học sinh, kể cả các học sinh chuyên Tin. Việc sử dụng thuật toán kết hợp với cấu trúc dữ liệu sẽ giúp việc cài đặt được dễ dàng hơn và độ phức tạp sẽ tối ưu hơn. Hiện nay, các tài liệu và các bài tập ứng dụng chưa được viết và tổng hợp lại để giúp giáo viên và học sinh đội tuyển có một tài liệu để nghiên cứu. Vì vậy tôi đã chọn chuyên đề Palindrome Tree để viết.

II. NỘI DUNG

II.1. Cây Palindrome

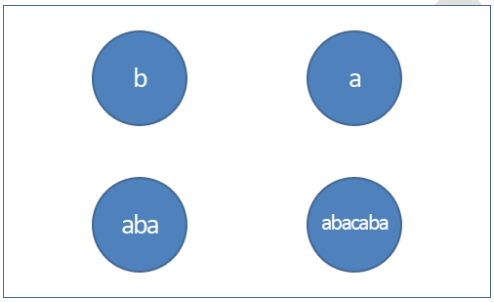
II.1.1. Khái niệm cây Palindrome

Cây Palindrome (hay còn được gọi là Eertree), được phát minh bởi Mikhail Rubinchik, là một loại cấu trúc dữ liệu hiệu quả được sử dụng để giải một số bài toán liên quan đến Palindrome.

Cây Palindrome là một cấu trúc dữ liệu tuyến tính mới, cho phép truy cập nhanh vào tất cả các xâu con Palindrome của một xâu hoặc một xâu các xâu. Cấu trúc này kế thừa một số ý tưởng từ việc xây dựng cả một cây hậu tố và cây hậu tố.

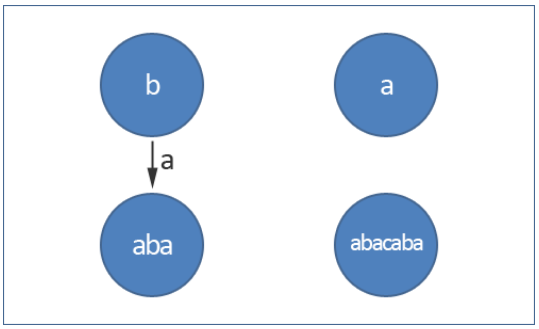
II.1.2. Cấu trúc của cây Palindrome

Như mọi loại cây khác, cây Palindrome cũng có các nút. Chẳng hạn như:



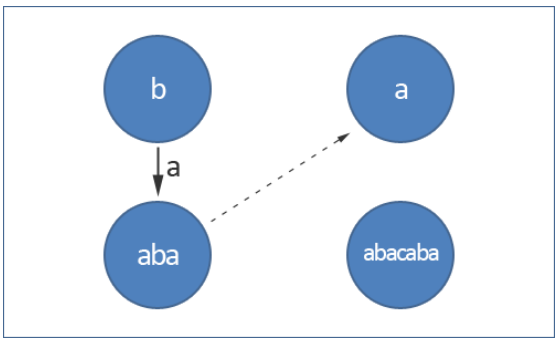
Hình 1: Ví dụ về các nút của một cây Palindrome

Ngoài nút ra cây còn có các cung để nối các nút. Cung nối giữa hai nút u và v được gán một chữ cái - ví dụ chữ X - nghĩa là ta có được Palindrome chứa ở nút v bằng cách thêm chữ X vào hai bên của Palindrome chứa ở nút u.



Hình 2: Xâu Palindrome *aba* có được bằng cách thêm chữ *a* vào 2 bên của xâu Palindome *b*

Cuối cùng, ta có thêm các liên kết hậu tố. Nút u có liên kết hậu tố đến nút w, nếu Palindrome chứa ở nút w là hậu tố không tầm thường lớn nhất của Palindrome chứa ở nút u. (hậu tố là một xâu con chứa các chữ cái cuối cùng của xâu, hậu tố không tầm thường (proper suffix) là hậu tố của một xâu và ngắn hơn xâu đó). Từ bây giờ ta sẽ gọi Palindrome lớn nhất mà là hậu tố không tầm thường của một xâu là Palindrome hậu tố lớn nhất của một xâu.



Hình 3. Trong ví dụ trên vì *a* là Palindrome hậu tố lớn nhất của *aba* nên có một liên kết hậu tố từ nút chứa xâu *aba* đến nút chứa xâu *a*.

Đặt tên cấu trúc dữ liệu này là cây Palindrome có vẻ không hợp lí lắm, vì nó có tận 2 gốc. Một sẽ chứa xâu Palindrome giả độ dài -1. Gốc này giúp ta cài đặt cây dễ dàng hơn, vì khi ta thêm hai chữ cái bất kì vào hai bên xâu độ dài -1 thì ta sẽ được xâu độ dài 1 và nó luôn là Palindrome. Gốc thứ hai chứa một xâu rỗng (xâu có độ dài 0), và xâu này cũng là Palindrome. Ta cho thêm một liên kết hậu tố từ hai gốc nối đến gốc chứa Palindrome độ dài -1.

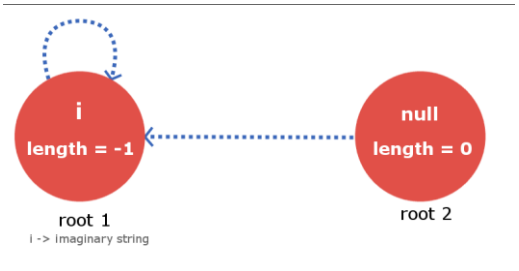
Lưu ý rằng ta không chứa xâu Palindrome vào nút khi cài đặt thực tế, nếu làm vậy ta sẽ tiêu tốn quá nhiều bộ nhớ. Nút thực tế sẽ chứa độ dài xâu Palindrome, chữ cái được gán vào các cung, và các liên kết hậu tố.

Các nút gốc và quy ước của chúng:

Cấu trúc dữ liệu cây / đồ thị này sẽ chứa **2 nút giả gốc**. Hơn nữa, chính xác coi nó là rễ của hai cây riêng biệt, được liên kết với nhau.

Root-1 sẽ là một nút giả sẽ mô tả một xâu có *độ dài = -1* (bạn có thể dễ dàng suy ra từ quan điểm thực hiện rằng tại sao chúng ta sử dụng như vậy). *Root-2* sẽ là một nút mô tả xâu null có *độ dài = 0* .

Root-1 có cạnh hậu tố được kết nối với chính nó (tự lặp) vì đối với bất kỳ xâu ảo nào có độ dài -1, hậu tố palindromic tối đa của nó cũng sẽ là tưởng tượng, vì vậy điều này là hợp lý. Bây giờ Root-2 cũng sẽ có cạnh hậu tố của nó được kết nối với Root-1 như đối với một xâu null (độ dài 0) không có xâu hậu tố palindromic thực sự có độ dài nhỏ hơn 0.



II.1.3. Ưu điểm của cây Palindrome

- Truy vấn và cập nhật trực tuyến
- Dễ để thực hiện
- Rất nhanh

II.2. Xây dựng cây Palindrome

II.2.1. Cách tạo một cây Palindrome

Để xây dựng Cây Palindrome, ta sẽ chỉ cần chèn từng ký tự vào trong xâu cho đến khi chúng ta kết thúc và khi ta chèn xong, khi đó ta sẽ có cây palindrome chứa tất cả các xâu palindrome riêng biệt của các xâu đã cho. Tất cả những gì chúng ta cần đảm bảo là, tại mỗi lần chèn một ký tự mới, cây palindrome phải duy trì tính năng ở trên. Nhiệm vụ đặt ra là làm thế nào để có thể hoàn thành nó.

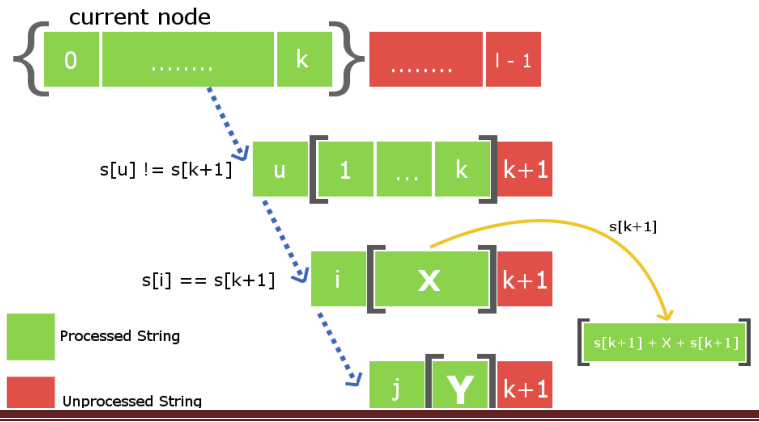
Giả sử ta được cung cấp một xâu **s** có độ dài **l** và ta đã chèn xâu đến chỉ số **k** (**k<l-1**). Bây giờ, ta cần chèn thêm ký tự **(k + 1)** . Chèn ký tự (**k + 1**) có nghĩa là chèn một nút mà Palindrome dài nhất kết thúc tại chỉ số (**k + 1**). Vì vậy, xâu palindrome dài nhất sẽ có dạng (**'s[k + 1]' + "[X]" + 's[k + 1]'**) và bản thân X sẽ là một palindrom. Thực tế là xâu X nằm tại chỉ số < k + 1 và là palindrome. Vì vậy, nó sẽ tồn tại trong cây palindrome vì ta đã duy trì thuộc tính rất cơ bản của nó rằng nó sẽ chứa tất cả các xâu con palindrome riêng biệt.

Vì vậy, để chèn ký tự **s[k + 1]**, ta chỉ cần tìm xâu X trong cây của mình và hướng cạnh chèn từ X với trọng số **s[k + 1]** vào một nút mới mà chứa **s[k + 1] + X + s[k+1]** . Công việc chính bây giờ là tìm xâu X trong thời gian hiệu quả. Như ta biết rằng ta đang lưu trữ liên kết hậu tố cho tất cả các nút. Do đó, để theo dõi nút với xâu X, ta chỉ cần di chuyển xuống liên kết hậu tố cho nút hiện tại, tức là nút có chứa s[k]. (Xem hình dưới đây để hiểu rõ hơn).

Nút hiện tại trong hình dưới cho biết đó là Palindrome lớn nhất kết thúc tại chỉ số k sau khi xử lý tất cả các chỉ số từ 0 đến k. Đường dẫn chấm màu xanh là liên kết của các cạnh hậu tố từ nút hiện tại đến các nút được xử lý khác trong cây. Xâu X sẽ tồn tại trong một trong các nút nằm trên xâu liên kết hậu tố này. Tất cả ta cần là tìm nó bằng cách lặp qua xâu xuống.

Để tìm nút cần thiết có chứa xâu X, ta sẽ đặt ký tự thứ **k + 1** ở cuối mỗi nút nằm trong chuỗi liên kết hậu tố và kiểm tra xem ký tự đầu tiên của chuỗi liên kết hậu tố tương ứng có bằng ký tự **k + 1** không.

Khi tìm thấy xâu X, ta chèn một cạnh với trọng số **s[k + 1]** và liên kết nó với nút mới chứa palindrome lớn nhất kết thúc tại chỉ số **k + 1**. Các phần tử mảng giữa các dấu ngoặc {} như mô tả trong hình bên dưới là các nút được lưu trữ trong cây.



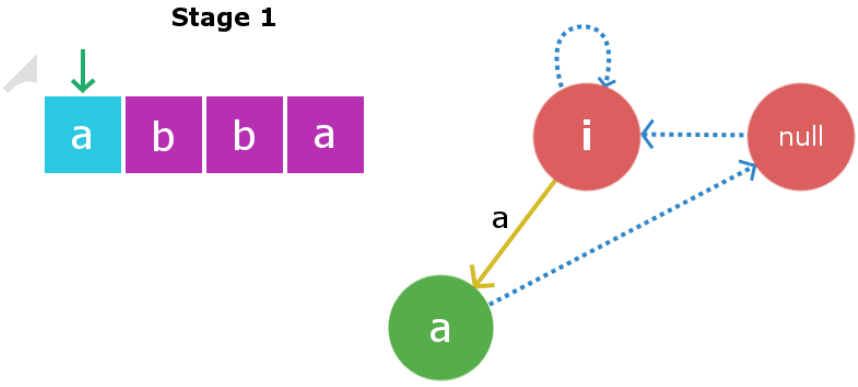
Vì chúng ta đã tạo một nút mới khi chèn [**k + 1**] này, do đó chúng ta cũng sẽ phải kết nối nó với con liên kết hậu tố của nó. Một lần nữa, để làm như vậy, ta sẽ sử dụng phép lặp liên kết hậu tố ở trên từ nút X để tìm một xâu **Y** mới sao cho **s[k + 1] + Y + s[k + 1]** là hậu tố palindrome lớn nhất cho nút mới tạo. Khi tìm thấy nó, ta sẽ kết nối liên kết hậu tố của nút mới được tạo với nút Y.

Lưu ý : Có hai khả năng khi ta tìm thấy chuỗi X. Khả năng đầu tiên là chuỗi **s[k+1]Xs[k+1]** không tồn tại trong cây và khả năng thứ hai là nếu nó đã tồn tại trong cây. Trong trường hợp đầu tiên, ta sẽ tiến hành theo cách tương tự nhưng trong trường hợp thứ hai, ta sẽ không tạo một nút mới riêng biệt mà sẽ chỉ liên kết cạnh chèn từ X đến nút **S[k + 1] + X + S[k + 1]** hiện có trong cây. Ta cũng không cần thêm liên kết hậu tố vì nút sẽ chứa liên kết hậu tố của nó.

Chẳng hạn xét xâu *s* = “**abba**” có độ dài = *4* .

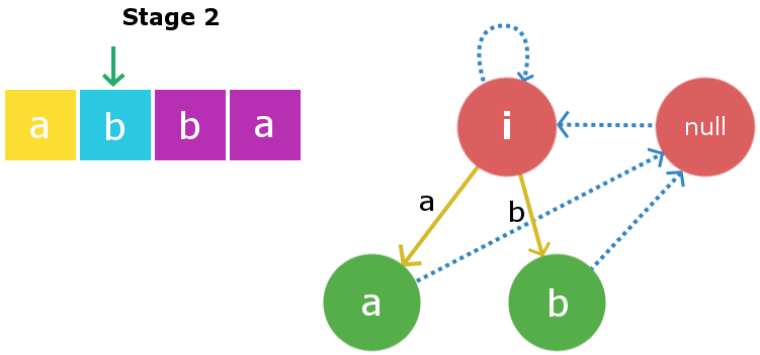
Ở trạng thái ban đầu, ta sẽ có hai nút gốc giả, một nút có độ dài -1 (một số xâu tưởng tượng **i**) và nút thứ hai là một xâu rỗng **null** có độ dài 0. Tại thời điểm này, ta chưa chèn bất kỳ kí tự nào vào cây. Root1 tức là nút gốc có độ dài -1 sẽ là nút hiện tại từ đó quá trình chèn diễn ra.

Giai đoạn 1: Ta sẽ chèn **s[0]** tức là '**a**'. Ta sẽ bắt đầu kiểm tra từ nút hiện tại tức là Root1. Chèn 'a' tại vị trí bắt đầu và kết thúc xâu có độ dài -1 sẽ cho một xâu có độ dài 1 và xâu này sẽ là “a”. Do đó, ta tạo một nút mới “a” và hướng của cạnh chèn từ root1 đến nút mới này. Bây giờ, xâu palindrome hậu tố lớn nhất cho xâu có độ dài 1 sẽ là một xâu rỗng, nên liên kết hậu tố của nó sẽ được chuyển đến root2 tức là xâu rỗng (null). Bây giờ, nút hiện tại sẽ là nút mới “a” này.

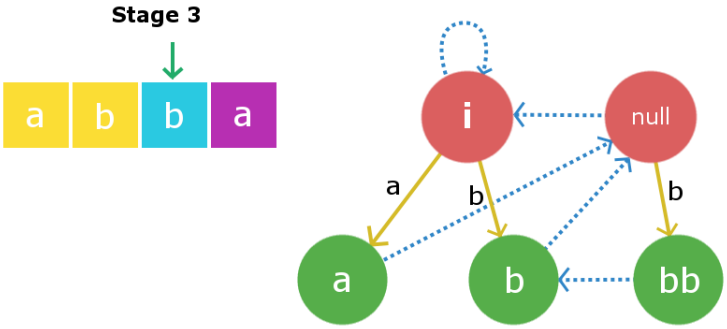


Giai đoạn 2: Ta sẽ chèn **s[1]** tức là '**b**'. Quá trình chèn sẽ bắt đầu từ nút hiện tại, tức là nút “a”. Ta sẽ duyệt qua chuỗi liên kết hậu tố bắt đầu từ nút hiện tại cho đến khi chúng ta tìm thấy xâu X phù hợp. Vì vậy ở đây đi qua liên kết hậu tố, ta lại tìm thấy root1 là xâu X. Một lần nữa chèn 'b' vào xâu có độ dài -1 sẽ cho xâu có độ dài 1 tức là xâu “b”. Liên

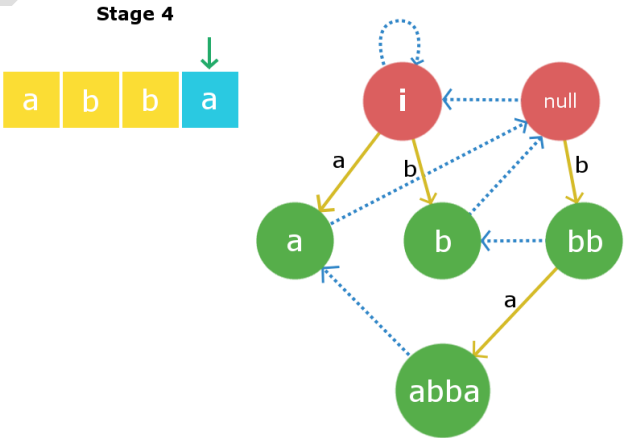
kết hậu tố cho nút này sẽ chuyển đến xâu rỗng (null) như được mô tả trong phần chèn ở trên. Bây giờ nút hiện tại sẽ là nút mới "b" này.



Giai đoạn 3: Ta sẽ chèn **s[2]** tức là '**b**'. Một lần nữa bắt đầu từ nút hiện tại, ta sẽ đi qua liên kết hậu tố của nó để tìm xâu X cần thiết. Trong trường hợp này, nó được tìm thấy là root2, tức là xâu rỗng (null) khi thêm 'b' vào đầu và cuối xâu null ta được một palindrome "bb" có độ dài 2. Do đó, ta sẽ tạo một nút mới "**bb**" và hướng của cạnh chèn từ xâu rỗng (null) đến xâu vừa được tạo. Bây giờ, palindrome hậu tố lớn nhất cho nút hiện tại này sẽ là nút "b". Vì vậy, ta sẽ liên kết cạnh hậu tố từ nút mới được tạo này với nút "b". Nút hiện tại bây giờ trở thành nút "bb".



Giai đoạn 4: Ta sẽ chèn **s[3]** tức là '**a**'. Quá trình chèn bắt đầu với nút hiện tại và trong trường hợp này, chính nút hiện tại là xâu X lớn nhất sao cho **s[0] + X + s[3]** là palindrome. Do đó, ta sẽ tạo một nút mới "**abba**" và liên kết cạnh chèn từ nút hiện tại "bb" với nút mới được tạo này bằng cạnh có trọng số 'a'. Bây giờ, hậu tố liên kết từ nút mới được tạo này sẽ được liên kết với nút "a" mà là Palindrome hậu tố lớn nhất.



II.2.2. Cài đặt xây dựng cây Palindrome trong ngôn ngữ lập trình C++

```
#include "bits/stdc++.h"
using namespace std;
#define MAXN  1000
struct Node
{
    // start và end là biến lưu chỉ số đầu và chỉ số cuối của một nút
    int start, end;
    // length: biến lưu độ dài của xâu con
    int length;
    // lưu trữ nút chèn cho tất cả các kí tự từ 'a' đến 'z'
    int insertEdg[26];
    // Lưu trữ nút Palindrome hậu tố lớn nhất cho nút hiện tại
    int suffixEdg;
};
// hai nút giả đặc biệt như đã giải thích ở trên
Node root1, root2;
// Lưu trữ thông tin nút để truy cập thời gian không thay đổi
Node tree[MAXN];
// Theo dõi nút hiện tại trong khi chèn
int currNode;
string s;
int ptr;
void insert(int idx)
{
//STEP 1//
    /* Tìm kiếm nút X sao cho s[idx] X S[idx] là Palindrome tối đa kết
   thúc tại vị trí  idx lặp lại liên kết hậu tố của nút hiện tại currNode
   để tìm X */
    int tmp = currNode;
    while (true)
    {
        int curLength = tree[tmp].length;
        if (idx - curLength >= 1 and s[idx] == s[idx-curLength-1])
            break;
        tmp = tree[tmp].suffixEdg;
    }
    /* Bắt đầu tìm X
     * Kiểm tra : if s[idx] X s[idx] tồn tại rồi hoặc chưa tồn tại*/
    if(tree[tmp].insertEdg[s[idx]-'a'] != 0)
    {
        // s[idx] X s[idx] already exists in the tree
        currNode = tree[tmp].insertEdg[s[idx]-'a'];
        return;
    }
```

```
        // tạo nút mới new Node
        ptr++;
        // tạo nút mới new Node như con của X với trọng số như s[idx]
        tree[tmp].insertEdg[s[idx]-'a'] = ptr;
        // tính độ dài length của nút mới  new Node
        tree[ptr].length = tree[tmp].length + 2;
        // cập nhật điểm cuối cho nút mới new Node
        tree[ptr].end = idx;
        // cập nhật điểm đầu cho nút mới new Node
        tree[ptr].start = idx - tree[ptr].length + 1;
    //STEP 2//
        /* Đặt cạnh hậu tố cho mới được tạo Node tree[ptr]. Tìm các xâu Y sao
cho
           s[idx] + Y + s[idx] là Palindrome hậu tố dài nhất có thể cho nút
        mới được tạo */
        tmp = tree[tmp].suffixEdg;
        // Tạo nút mới như nút hiện tại currNode
        currNode = ptr;
        if (tree[currNode].length == 1)
        {
            // Nếu độ dài xâu Palindrome mới bằng 1
            // thì làm cho liên kết hậu tố của nó là xâu rỗng
            tree[currNode].suffixEdg = 2;
            return;
        }
        while (true)
        {
            int curLength = tree[tmp].length;
            if (idx-curLength >= 1 and s[idx] == s[idx-curLength-1])
                break;
            tmp = tree[tmp].suffixEdg;
        }
        // Tìm xâu Y
        // Liên kết các nút liên kết hậu tố hiện tại với s[idx]+Y+s[idx]
        tree[currNode].suffixEdg = tree[tmp].insertEdg[s[idx]-'a'];
    }
    // driver program
    int main()
    {
        // Khởi tạo cây
        root1.length = -1;
        root1.suffixEdg = 1;
        root2.length = 0;
        root2.suffixEdg = 1;
        tree[1] = root1;
```

```
        tree[2] = root2;
        ptr = 2;
        currNode = 1;
        // cho xâu
        s = "abcbab";
        int l = s.length();
        for (int i=0; i<l; i++)
            insert(i);
        // In tất cả các xâu con Palindrome khác nhau
        cout << "All distinct palindromic substring for "
             << s << " : \n";
        for (int i=3; i<=ptr; i++)
        {
            cout << i-2 << ") ";
            for (int j=tree[i].start; j<=tree[i].end; j++)
                cout << s[j];
            cout << endl;
        }
        return 0;
}
```

Nhận xét:

Trong quá trình xây dựng cây Palindrome cho một xâu độ dài *n*. Ta thấy rằng khi ta xử lí từng chữ cái một, đầu của liên kết hậu tố Palindrome lớn nhất của tiền tố được xử lí luôn di chuyển sang bên phải. Do đó, độ phức tạp của việc xây dựng cây Palindrome là *O(n)*.

Ứng dụng:

- Đếm số lượng Palindrome xuất hiện thêm

Bài toán: Cho thêm chữ cái *x* vào cuối xâu *S*, đếm số lượng Palindrome xuất hiện thêm trong xâu *S*. Ví dụ khi ta cho thêm chữ cái *a* vào cuối xâu *aba*, ta có thêm một Palindrome nữa là *a*.

Lời giải khá là rõ ràng: Ta xây dựng cây Palindrome cho xâu *S* ban đầu, và với mỗi chữ cái mới thêm vào, ta biết được số Palindrome mới xuất hiện thêm bằng cách đếm số nút vừa được tạo ra trên cây Palindrome. Lưu ý: số Palindrome xuất hiện thêm sau khi thêm một chữ cái vào một xâu bằng 1 hoặc bằng 0.

- Đếm số lượng xâu con liên tiếp là Palindrome

- Đếm số lần xuất hiện của Palindrome trong xâu

II.3. Một số bài toán ứng dụng

II.3.1. Bài toán đếm số lượng xâu con liên tiếp là Palindrome

Bài toán:

Một xâu được gọi là Palindrome nếu xâu đó đọc từ trái sang phải cũng giống như đọc từ phải sang trái. Ví dụ: xâu abba, ata là các xâu Palindrome.

Yêu cầu: Cho trước một xâu *s*, xâu con của nó là một xâu các kí tự liên tiếp nhau. Hãy xác định xem có bao nhiêu xâu con liên tiếp là Palindrome trong xâu đã cho.

Dữ liệu: Vào từ file **SUBPAL.INP**

Gồm duy nhất một dòng chứa xâu *s* chỉ chứa các chữ cái la tinh thường, độ dài của xâu không quá 10^5 kí tự.

Kết quả: Ghi ra file **SUBPAL.OUT**

In ra duy nhất một dòng chứa số lượng xâu con liên tiếp là Palindrome

Ví dụ:

SUBPAL.INP	SUBPAL.OUT
aba	4

Ràng buộc:

- ✓ Xâu *s* chỉ bao gồm các chữ cái latinh in thường.
- ✓ Sub1: 30% test có $1 \le |s| \le 100$.
- ✓ Sub2: 30% test có $1 \le |s| \le 1000$.
- ✓ Sub3: 40% test có $1 \le |s| \le 10^5$.

❖ **Xác định bài toán:**

Input: xâu *s*

Output: Số lượng xâu con liên tiếp là Palindrome

❖ **Phân tích thuật toán:**

CÁCH 1: Đây là một bài toán về xâu Palindrome rất quen thuộc. Nếu làm theo cách bình thường chúng ta sử dụng thuật toán quy hoạch động để kiểm tra xem một xâu con từ vị trí *i* đến vị trí *j* $s_i, s_{i+1}, \ldots, s_j$ có là xâu con Palindrome hay không và dùng mảng *c[i][j]* để đánh dấu, như vậy độ phức tạp thuật toán sẽ là là $O(n^2)$.

CÁCH 2: Sử dụng thuật toán Manacher

Ta thấy dữ liệu đề bài cho là độ dài xâu $\le 10^5$, nên ta không thể sử dụng phương pháp đơn giản trên, mà phải sử dụng thuật toán khác. Một trong các thuật toán đó là Manacher với độ phức tạp *O(n)*.

CÁCH 3: Sử dụng Palindrome Tree

Sử dụng cấu trúc cây Palindrome với độ phức tạp *O(n)* vì nó có thể áp dụng được cho nhiều bài toán khác.

-Xây dựng cây Palindrome, tại mỗi bước ta chèn thêm vào một nút có giá trị là xâu con palindrome.

- Bắt đầu từ xâu con Palindrome có độ dài 1, tức là một kí tự.

❖ **Cài đặt chương trình:**

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

```
const int MAXN = 105000;
struct node {
        int next[26];
        int len;
        int sufflink;
        int num;
    };
int len;
char s[MAXN];
node tree[MAXN];
int num;// node 1 - gốc với độ dài len-1, node 2-gốc với độ dài
len0
int suff;           // max suffix palindrome
long long ans;
bool addLetter(int pos) {
        int cur = suff, curlen = 0;
        int let = s[pos] - 'a';
        while (true) {
            curlen = tree[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
s[pos])
                break;
            cur = tree[cur].sufflink;
        }
        if (tree[cur].next[let]) {
            suff = tree[cur].next[let];
            return false;
        }
        num++;
        suff = num;
        tree[num].len = tree[cur].len + 2;
        tree[cur].next[let] = num;
        if (tree[num].len == 1) {
            tree[num].sufflink = 2;
            tree[num].num = 1;
            return true;
        }
        while (true) {
```

```
            cur = tree[cur].sufflink;
            curlen = tree[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
s[pos]) {
                tree[num].sufflink = tree[cur].next[let];
                break;
            }
        }
        tree[num].num = 1 + tree[tree[num].sufflink].num;
        return true;
    }
    void initTree() {
        num = 2; suff = 2;
        tree[1].len = -1; tree[1].sufflink = 1;
        tree[2].len = 0; tree[2].sufflink = 1;
    }
    int main() {
        freopen("SUBPAL.INP", "r", stdin);
        freopen("SUBPAL.OUT", "w", stdout);
        gets(s);
        len = strlen(s);
        initTree();
        for (int i = 0; i < len; i++) {
            addLetter(i);
            ans += tree[suff].num;
        }
        cout << ans << endl;
        return 0;
    }
```

- **Nhận xét độ phức tạp:** Chủ yếu thời gian của chương trình phụ thuộc vào thời gian xây dựng cây Palindrome cho xâu kí tự ban đầu với độ dài *n*, nên độ phức tạp là: *O(n)* => độ phức tạp tuyến tính.
- **Test:**
  https://drive.google.com/file/d/1YLlmISXbXrbRPUfuy8HqD7DLvcfwtAgI/view?usp=sharing
- **Cảm nhận:**

Đây là một dạng bài toán phát triển từ bài toán cơ bản về xâu Palindrome. Từ xâu ban đầu ta đếm số xâu con liên tiếp là Palindrome. Kết qur độ dài các xâu con được lưu vào mỗi nút trong cây. Tổng các nút là kết quả cần tìm.

II.3.2. Bài toán Palindromeness

(Nguồn Codechef - Palindromeness)

Bài toán:

Chúng ta định nghĩa Palindomeness của một xâu bằng cách sau:

- Nếu xâu không phải là một Palindrome thì Palindomeness của nó bằng 0.
- Palindomeness của một xâu chỉ gồm một chữ cái bằng 1.
- Palindomeness của một xâu *S* có độ dài lớn hơn một là 1 + "Palindomeness của xâu được hình thành bởi các kí hiệu đầu tiên [|*S*|/2] của xâu *S*".

Chẳng hạn như xét các ví dụ sau:

- Palindomeness của xâu *zxqfd* bằng 0 vì xâu này không phải là xâu Palindome.
- Palindomeness của xâu *a* bằng 1, theo định nghĩa.
- Palindomeness của xâu *aa* bằng 2, vì để có "*aa"* = 1 + Palindomeness của xâu "*a*" (mà bằng 1 theo định nghĩa). Vì vậy kết quả bằng 2.
- Palindomeness của xâu abacaba bằng 3 (giải thích tương tự).

Yêu cầu: Bạn được cho một xâu *S*. Tìm tổng của tất cả các Palindomeness của tất cả các xâu con khác rỗng của *S* (tức là *S*[*i..j*], trong đó $i \leq j$). Nói cách khác, bạn phải tính tổng của các Palindomeness của $N * (N + 1) / 2$ xâu con của *S*, với *N* chính là độ dài của xâu *S*.

Dữ liệu: Vào từ file **PALPROB.INP**

- Dòng đầu tiên chứa số nguyên *T* là số lượng test.
- *T* dòng tiếp theo, mỗi dòng chưa một xâu *S* mô tả một test tương ứng.

Kết quả: Ghi ra file **PALPROB.OUT**

- Gồm *T* dòng, mỗi dòng ghi ra một số nguyên là kết quả tìm được của test tương ứng trong dữ liệu vào.

Ví dụ:

PALPROB.INP	PALPROB.OUT
2 zxqfd aba	5 5

Giải thích:

- Trong ví dụ 1: Không có xâu Palindrome nào là xâu con của xâu đã cho mà có độ dài lớn hơn 1. Mỗi kí tự riêng lẻ là một Palindrome nên Palindromeness của mỗi kí tự riêng lẻ bằng 1.

- Trong ví dụ 2: Palindromeness của xâu *aba* là bằng 2 và tổng các Palindromeness của mỗi kí tự riêng lẻ là bằng 3.

Ràng buộc:

- ✓ $1 \leq T \leq 3$
- ✓ Xâu S chỉ bao gồm các chữ cái latinh in thường.
- ✓ Sub1: 40% test có $1 \leq |S| \leq 100$.
- ✓ Sub2: 35% test có $1 \leq |S| \leq 1000$.
- ✓ Sub3: 25% test có $1 \leq |S| \leq 10^5$.

❖ **Xác định bài toán:**

Input: - Số lượng test T

- Xâu gồm các chữ cái latinh thường có độ dài $\leq 10^5$

Output: T dòng, mỗi dòng là tổng của các Palindomeness của $N * (N + 1)/2$ xâu con của S, với N là độ dài của xâu S.

❖ **Phân tích thuật toán:**

Vấn đề của bài toán là phải tìm ra tất cả các Palindromes khác nhau, số lượng của chúng và Palindromeness của chúng.

Trước hết, ta thấy: số lượng palindrome khác nhau trong một xâu có độ dài n tối đa là n (để chứng minh điều này chú ý đến số lượng Palindrome khác nhau có thể kết thúc tại bất kỳ vị trí i nào).

Để tính số lượng của tất cả các Palindrome khác nhau, ta sẽ xây dựng một cây Palindrome của xâu s đã cho. Tóm lại, cây Palindrome là một cây có các đỉnh biểu thị các Palindrome và tồn tại một cạnh có hướng từ u đến v nếu $v = xux$ đối với một ký tự x là Palindrome, v được tạo thành bằng cách thêm x vào Palindrome u . Ngoài các cạnh bình thường, chúng ta có các liên kết hậu tố từ u đến v nếu v là Palindrome hậu tố dài nhất của u .

Ngoài thông tin thường được lưu trữ trong một đỉnh của cây Palindrome, ta sẽ lưu trữ một *số lượng* biến lưu trữ chỉ số, trong đó Palindrome liên kết với đỉnh này là Palindrome dài nhất và cũng là hậu tố ngược. Để tính số lượng, ta theo dõi Palindrome dài nhất khi duyệt xâu. Tức là cứ khi nào chúng ta thêm một ký tự mới, ta kiểm tra xem Palindrome hậu tố dài nhất đã có trong cây chưa? Khi đó, ta sẽ tăng *số đếm* của nó còn không sẽ tạo một đỉnh mới và khởi tạo *số đếm* của nó bằng 1. Liên kết hậu tố ngược cũng rất dễ dàng, bất cứ khi nào ta thêm một liên kết hậu tố, thì cũng thêm một liên kết hậu tố ngược. Bằng cách này, ta sẽ xây dựng cây Palindrome.

✓ **Tính số lượng của tất cả các palindromes**

Tại mỗi đỉnh, ta có số lượng các chỉ số mà trong đó đây là palindrome dài nhất. Nhưng Palindrome t này cũng có thể kết thúc tại các chỉ số, trong đó một số Palindrome khác dài nhất và Palindrome t là hậu tố của nó. Đó là, đỉnh này có thể truy cập được bởi một số đỉnh thông qua đường dẫn hậu tố. Bây giờ để tìm tất cả các lần xuất

hiện của *t*, chúng ta sẽ chạy một dfs (đó là lý do tại sao ta cũng đã lưu trữ hậu tố ngược) trên cây, chỉ xử lý hậu tố ngược như các cạnh và ở mỗi đỉnh ta sẽ thêm biến đếm của tất cả các con của nó. Vì chúng ta có tất cả sự xuất hiện của Palindromes của các con của nó (vì dfs đã chạy trên chúng) nên giờ chúng ta có tất cả các chỉ số trong đó *t* không phải là Palindrome dài nhất mà nó là hậu tố của Palindrome dài nhất. Thêm phần này vào biến đếm *của* đỉnh *t*.

- ✓ **Tính toán Palindromeness**

Để tính toán Palindromeness, c ta lại chạy một dfs (nguyên vẹn hai dfs có thể được hợp nhất với nhau) trên cây Palindrome coi hậu tố ngược là các cạnh. Trong khi di chuyển cây theo thứ tự dfs, ta sẽ duy trì một mảng *P* [] lưu trữ phần trên của tất cả các tiền tố của xâu tương ứng với đỉnh mà chúng ta hiện đang xử lý.

❖ **Cài đặt chương trình:**

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5 + 42, sigma = 26;
int len[maxn], link[maxn], hlink[maxn];
int to[maxn][sigma], cnt[maxn];
int s[maxn], dp[maxn];
int sz, last, n;
void init()
{
    memset(cnt, 0, sizeof(cnt));
    memset(to, 0, sizeof(to));
    memset(dp, 0, sizeof(dp));
    link[0] = hlink[0] = 1;
    n = last = 0;
    len[1] = -1;
    s[n++] = -1;
    sz = 2;
}
int get_link(int v)
{
    while(s[n - 1] != s[n - len[v] - 2])
        v = link[v];
    return v;
}
void add_letter(char c)
{
    s[n++] = c -= 'a';
    last = get_link(last);
    if(!to[last][c])
{
        len[sz] = len[last] + 2;
```

```
            link[sz] = to[get_link(link[last])][c];
            hlink[sz] = to[get_link(hlink[last])][c];
            while(len[hlink[sz]] * 2 > len[sz])
                hlink[sz] = link[hlink[sz]];
            dp[sz] = 1 + dp[hlink[sz]] * (len[hlink[sz]] == len[sz] / 2);
            to[last][c] = sz++;
        }
        last = to[last][c];
        cnt[last]++;
    }
    void solve()
    {
        init();
        string s;
        cin >> s;
        for(auto c: s)
            add_letter(c);
        int64_t ans = 0;
        for(int i = sz - 1; i > 1; i--)
        {
            cnt[link[i]] += cnt[i];
            ans += cnt[i] * dp[i];
        }
        cout << ans << "\n";
    }
    int main()
    {
        ios::sync_with_stdio(0);
        cin.tie(0);
        int t;
      freopen("PALPROB.INP", "r", stdin);
      freopen("PALPROB.OUT", "w", stdout);
        cin >> t;
        while(t--)
            solve();}
```

- **Nhận xét độ phức tạp:** Chủ yếu thời gian của chương trình phụ thuộc vào thời gian xây dựng cây Palindrome và thời gian của DFS cập nhật mảng P[] qua đoạn code:

```
void dfs(int u)
{
    int i, v;
    if( T.len == 1 ) P[1] = 1;
    else P[T.len] = P[T.len/2] + 1;
    T.palindromness = P[T.len];
    for(i = 0; i< T.revSuffixLink.size(); i++)
```

```
{
        v = T.revSuffixLink[i];
        dfs(v);
        T.count += T[v].count;
    }
    P[T.len] = 0;
}
```

Nên độ phức tạp là: *O(n.T).*

ĐỘ PHỨC TẠP: *O(n.T).*

❖ **Test:**

https://drive.google.com/file/d/1A9_MIKncToszLBdDT0bNR9pW8JkaL-Pq/view?usp=sharing

❖ **Cảm nhận:**

Đây là bài toán thuộc lớp bài toán ứng dụng Palindrome Tree. Bài toán giúp ta biết cách lưu trữ hậu tố ngược trên cây.

II.3.3. Bài toán Palisection

(Nguồn http://codeforces.com/contest/17/problem/E)

Bài toán:

Trong một lớp học tiếng Anh, Nick không có gì để làm cả, và cậu nhớ về những xâu tuyệt vời gọi là các palindrome. Biết một xâu được gọi là palindrome nếu nó có thể được đọc theo cùng một cách cả từ trái sang phải và từ phải sang trái. Chẳng hạn như các xâu sau: “eye”, “pop”, “level”, “aba”, “deed”, “racecar”, “rotor”, “madam”.

Nick bắt đầu xem xét cẩn thận tất cả các palindromes trong văn bản mà mọi người đang đọc trong lớp. Đối với mỗi lần xuất hiện của mỗi palindrome trong văn bản, cậu đã viết một cặp - vị trí bắt đầu và vị trí kết thúc của sự xuất hiện này trong văn bản. Nick gọi mỗi lần xuất hiện của mỗi palindrome mà cậu tìm thấy trong văn bản là palindrome con . Khi tìm thấy tất cả các palindrome con, cậu quyết định tìm hiểu xem có bao nhiêu cặp khác nhau trong số các palindrome con này giao nhau. Hai palindrome con là giao nhau nếu chúng có chung các vị trí trong văn bản. Không có palindrome nào giao nhau với chính nó.

Chẳng hạn với xâu văn bản “babb”, Nick sẽ thực hiện như sau:

Đầu tiên, cậu viết ra tất cả các palindrome con:

- ✓ “b” _ 1..1
- ✓ “bab” _ 1..3
- ✓ “a” _ 2..2
- ✓ “b” _ 3..3
- ✓ “bb”_ 3..4
- ✓ “b” _ 4..4

Sau đó, Nick đếm số lượng các cặp khác nhau trong số các palindrome con giao nhau. Có sáu cặp như vậy:

1) 1..1 giao nhau với 1..3
2) 1..3 giao nhau với 2..2
3) 1..3 giao nhau với 3..3
4) 1..3 giao nhau với 3..4
5) 3..3 giao nhau với 3..4
6) 3..4 giao nhau với 4..4

Vì thao tác các bước này bằng tay rất mệt mỏi nên Nick đã yêu cầu bạn giúp cậu ấy viết chương trình tìm số lượng các cặp palindrome con khác nhau mà giao nhau. Hai cặp palindrome con được coi là khác nhau nếu một trong hai cặp chứa một palindrome con mà cặp kia không có.

Dữ liệu: Vào từ file văn bản **PALISECT.INP** gồm:

- Dòng đầu tiên chứa số nguyên dương *n* là độ dài của xâu.
- Dòng thứ hai chứa xâu *s* gồm n chữ cái latinh thường.

Kết quả: Ghi ra file văn bản **PALISECT.OUT** gồm duy nhất một dòng ghi một số nguyên là số cặp Palindrome con khác nhau mà giao nhau. Kết quả modul cho 51123987.

Ví dụ:

PALISECT.INP	PALISECT.OUT
4 babb	6
2 aa	2

Ràng buộc:

- ✓ $1 \leq n \leq 2.10^6$
- ✓ Xâu *S* chỉ bao gồm các chữ cái latinh in thường.
- ✓ Sub1: 40% test có $1 \leq n \leq 100$.
- ✓ Sub2: 30% test có $n \leq 1000$.
- ✓ Sub3: 30% test có $n \leq 2.10^6$.

❖ **Xác định bài toán:**

Input: - Số *n* nguyên dương, $n \leq 10^6$

- Xâu *s* gồm *n* kí tự latinh thường.

Output: số dư của số lượng cặp Palindrome con khác nhau mà giao nhau chia cho 51123987.

❖ **Phân tích thuật toán:**

CÁCH 1: Sử dụng thuật toán Manacher để tìm các *l[i], r[i]* với độ phức tạp *O(n).*

=> Độ phức tạp tuyến tính.

Cài đặt chương trình

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define P 51123987
#define maxn 4000005
 int N,M=1,f[maxn],l[maxn],r[maxn],ans;
char a[maxn],b[maxn];
 int main()
{
      ios_base::sync_with_stdio(0);
      cin.tie(0);
      cout.tie(0);
     freopen("PALISECT.INP", "r", stdin);
      freopen("PALISECT.OUT", "w", stdout);
    cin>>N>>a;
    b[0]=1;
    for (int i=0; i<N; i++) b[++M]=a[i],M++;
    for (int i=1,p=0,q=0; i<=M; i++)
    {
        f[i]=q>i?std::min(f[2*p-i],q-i):1;
        for (;b[i+f[i]]==b[i-f[i]]; f[i]++);
        if (i+f[i]>q) p=i,q=i+f[i];
    }
    for (int i=1; i<=M; i++) l[i-f[i]+1]++,l[i+1]--,r[i]++,r[i+f[i]]--
,(ans+=f[i]/2)%=P;
    ans=1ll*ans*(ans-1)/2%P;
    for (int i=1,s=0; i<=M; i++)
    {
        l[i]+=l[i-1],r[i]+=r[i-1];
        if (i%2==0) (ans-=1ll*s*l[i]%P)%=P,(s+=r[i])%=P;
    }
    cout<<((ans+P)%P)<<"\n";
}
```

CÁCH 2:

Sử dụng cấu trúc dữ liệu cây Palindrome

Cài đặt chương trình:

```
#include<bits/stdc++.h>
#define ll long long
#define pb push_back
#define MAX 1e18
#define MIN -1e18
#define MOD 51123987
//#define mod2 20071027
//#define MOD 998244353
#define base 139
//#define mod 1000074259
#define base2 31
#define memz(a) memset(a, 0, sizeof(a))
#define memn(a) memset(a, -1, sizeof(a))
#define in1(a) scanf("%lld", &a)
```

```
#define in2(a, b) scanf("%lld%lld", &a, &b)
#define TC(c) printf("Case #%d: ", ++c)
#define out(x) cout << #x << " -> " << x << endl;
#define FAST ios_base::sync_with_stdio(false); cin.tie(NULL);
#define FILE   freopen("input.txt", "r", stdin);  freopen("out.txt", "w",
stdout);
using namespace std;
const int N = 2000005;
int avail, sz, tc;
int len[N], link[N], t, cur, node[N],occ[N],  a[N], tocc[N];
// ll  b[N];
string s, r; // 1-indexed
vector<pair<int, int>>tree[N];
void init()
{
    for(int i=0; i<N-3; i++)
    {
        tree[i].clear();
    }
    memz(occ);
    memz(tocc);
    len[1] = -1, link[1] = 1;
    len[2] = 0, link[2] = 1;
    avail = cur = 2;
    s = "$" + s;
}
void extend(int pos)
{
    while (s[pos - len[cur] - 1] != s[pos]) cur = link[cur];
    int x = link[cur], c = s[pos] - 'a';
    while (s[pos - len[x] - 1] != s[pos]) x = link[x];
    int ok = 0;
    for (auto u : tree[cur])
    {
        if (u.second == c)
            ok = u.first;
    }
    if (!ok)
    {
        tree[cur].pb({++avail, c});
        len[avail] = len[cur] + 2;
        if (len[avail] == 1)
            link[avail] =  2 ;
        else
        {
            for(auto u:tree[x])
            {
                if(u.second==c)
                {
                    ok=u.first;
                }
            }
            link[avail]=ok;
        }
        cur = avail;
    }
    else cur = ok;
    tocc[cur]++;
    node[pos] = cur;
}
int dfs(int x)
{
    if (x < 3) return 0;
```

```
    if (occ[x]) return occ[x];
    return occ[x] = 1 + dfs(link[x]);
}
void pl_tree()
{
    init();
    for (int i = 1; i <= sz; i++)
    {
        extend(i);
    }
    for (int i = 3; i <= avail; i++)
        if (occ[i] == 0)
        {
            occ[i] = dfs(i);
        }
}
int solve()
{
    cin >> sz >> s;
    r = s;
    ll res = 0;
    pl_tree();
    for (int i = 1; i <= sz; i++)
        a[i] = occ[node[i]];
    reverse(r.begin(), r.end());
    s = r;
    pl_tree();
    ll b=0;
    for (int i = 1; i < sz; i++)
    {
        b = (b + occ[node[i]])%MOD;
        res = (res+(b*a[sz-i])%MOD)%MOD;
    }
    ll tot = 0;
    for (int i = avail; i > 2; i--)
    {
        tocc[link[i]] += tocc[i];
        tot += (ll)tocc[i];
    }
    ll x=tot, y=tot-1;
    if(y%2==0) y/=2;
    else x/=2;
    tot = ((x%MOD)*(y%MOD))%MOD;
    // for (int i = 1; i < sz; i++) {
    //      res = (res+(a[i] * b[sz - i])%MOD)%MOD;
    // }
    ll fin = (tot-res)%MOD;
    if(fin<0) fin+=MOD;
    cout<<fin;
    //printf("%lld\n", fin);
    return 0;
}
int main()
{
    freopen("PALISECT.INP","r",stdin);
    freopen("PALISECT.OUT","w",stdout);
    return solve();
}
```

- **Nhận xét độ phức tạp:** Chủ yếu thời gian của chương trình phụ thuộc vào xây dựng cây Palindrome, nên độ phức tạp là: *O(2*n)*

ĐỘ PHỨC TẠP: Tuyến tính.

❖ **Test:**

https://drive.google.com/file/d/1wWQnznWHTtokVXCdP4z23ZcvtTras6LM/view?usp=sharing

❖ **Cảm nhận:**

Đây là bài toán phát triển từ bài toán xâu Palindrome cơ bản là đếm số lượng cặp Palindrome khác nhau. Sau đó tìm khoảng giao nhau của các cặp này.

II.3.4. Bài toán Virus synthesis

(Nguồn: CERC 14 - Bài G - Virus synthesis)

Bài toán:

Virus thường có hại cho sức khỏe của bạn. Làm thế nào để tổng hợp được virus tốt để có thể chiến đấu với các loại virus khác? Chính vì vậy, bạn cần tìm ra cách tổng hợp những virus tốt như vậy.

Bạn được cho trước một xâu gồm các chữ cái A, G, T và C. Chúng tương ứng với trình tự nucleotide DNA của virus mà bạn muốn tổng hợp, được sử dụng các thao tác sau:

• Thêm một nucleotide vào đầu hoặc cuối xâu hiện có.

• Sao chép xâu, đảo ngược đoạn đã sao chép và dán nó vào đầu hoặc cuối xâu ban đầu (ví dụ: AGTC có thể trở thành AGTCCTGA hoặc CTGAAGTC).

Có rất nhiều trình tự như vậy, một trong số chúng là rất dài. Vì vậy vấn đề cần quan tâm đến là hiệu quả?

Yêu cầu: Hãy tìm cách tổng hợp virus sao cho số thao tác được sử dụng là nhỏ nhất.

Dữ liệu: Vào từ file văn bản **VIRUS.INP**:

- Dòng đầu tiên chứa số lượng test *T*.
- *T* dòng tiếp theo, mỗi dòng chứa một xâu kí tự khác rỗng có độ dài không lớn hơn 10^5 kí tự, chỉ bao gồm các chữ in hoa A, C, G và T.

Kết quả: Ghi ra file văn bản **VIRUS.OUT:**

- Gồm T dòng, mỗi dòng ghi một số nguyên là số thao tác nhỏ nhất được sử dụng tương ứng với xâu trong file dữ liệu vào.

Ví dụ:

VIRUS.INP	VIRUS.OUT
4 AAAA AGCTTGCA AAGGGGAAGGGGAA AAACAGTCCTGACAAAAAAAAAAAAC	3 8 6 18

Giới hạn:

 Có 20% số test ứng với 20% số điểm có $T \leq 10$; độ dài của xâu ban đầu ≤ 100

 Có 40% số test ứng với 40% số điểm có $T \le 10$; độ dài của xâu ban đầu $\le 10^3$

 Có 40% số test còn lại ứng với 40% số điểm có $T \le 100$; độ dài của xâu ban đầu $\le 10^5$.

❖ **Xác định bài toán:**

Input: Số lượng test *T*; Các xâu ban đầu *(*$T \le 100$, độ dài của xâu ban đầu $\le 10^5$)

Output: *T* số nguyên, mỗi số nguyên là số lượng thao tác nhỏ nhất được sử dụng để tổng hợp virus tương ứng với xâu ban đầu.

❖ **Phân tích thuật toán:**

CÁCH 1: Sử dụng thuật toán Manacher

Độ phức tạp là *O(nlogn)*

CÁCH 2: Sử dụng cấu trúc dữ liệu Palindrome Tree

Đây là một bài toán liên quan đến tìm tất cả các Palindrome có thể trong xâu ban đầu.

Chẳng hạn với xâu TGGTTAGGATTA có các xâu Palindrome: TT, GG, TGGT, AGGA, TAGGAT, TTAGGATT, ATTA.

Giả sử ta liệt kê được tất cả các Palindrome có độ dài > 1 lần lượt là*:* $p_1, p_2, ..., p_s$. Với mỗi p_i ta ghi nhớ trim (p_i) – Palindrome p_i không chứa kí tự đầu và kí tự cuối. Sau đó ta tính chi phí tối thiểu để tổng hợp các xâu Palindrome $p_1, p_2, ..., p_s$ theo thứ tự tăng dần của độ dài xâu.

Với mỗi Palindrome p_i ta tính:

- ✓ full[i] – chi phí để tạo p_i.
- ✓ half [i] – chi phí để tạo một nửa của p_i.

Khi ta biết chi phí tổng hợp tất cả các Palindrome trong từ thì rất đơn giản ta có thể tính chi phí của toàn bộ từ bằng cách kiểm tra tất cả => Độ phức tạp là *O(nlogn)*

❖ **Cài đặt chương trình:**

CÁCH 1 :

```
#include <bits/stdc++.h>
#define dprintf(...)
//#define dprintf(...) fprintf(stderr,__VA_ARGS__)
using namespace std;
const int maxn = 100*1000;
const int maxp = 3*maxn+1;

int pal[maxn];
int who[maxn];
int middle[maxp];
int radius[maxp];
vector<int> link[maxp];
vector<int> endq[maxn];
vector<int> events[maxn];
int shorter[maxp];
int half[maxp];
int dp[maxp],dph[maxp];
string A;
int pc;
int n;
int trim(int node, int x, int s=0)
{
```

```
    if (x==0)
        return node;
    if ( x%(1<<(s+1)) != 0)
    {
        node = link[node][s];
        x -= (1<<s);
    }
    return trim(node,x,s+1);
}
void manacher()
{
    int f = 0;
    pal[0] = 0;
    who[0] = 0;
    middle[0] = 0;
    radius[0] = 0;
    pc = 1;
    for(int i=1; i<n; i++)
    {
        who[i] = 0;
        if (f+pal[f]>i)
        {
            int j = f - (i - f);
            who[i] = who[j];
            pal[i] = pal[j];
            int over = i+pal[i]-f-pal[f];
            if (over>0)
            {
                pal[i] -= over;
                who[i] = trim(who[i],over);
            }
        }
        else
            pal[i] = 0;
        while(i+pal[i]<n && i-pal[i]>0 && A[i+pal[i]]==A[i-pal[i]-1])
        {
            pal[i]++;
            f = i;
            middle[pc] = i;
            radius[pc] = pal[i];
            link[pc].clear();
            link[pc].push_back(who[i]);
            int q = 1;
            while((1 << q) <= radius[pc])
            {
                link[pc].push_back(link[link[pc][q-1]][q-1]);
                q++;
            }
            who[i] = pc;
            pc++;
        }
    }
}

int query(int i, set<int> &S, int x)
{
    auto t = S.lower_bound(x+1);
    if (t!=S.begin())
    {
        t--;
        int over = i - (*t - pal[*t]);
        return trim(who[*t], over);
    }
```

```
    else
        return 0;
}

int main()
{
  freopen("VIRUS.INP", "r", stdin);
   freopen("VIRUS.OUT", "w", stdout);
    int TT;
    cin >> TT;
    while(TT--)
    {
        cin >> A;
        n = A.length();
        manacher();
        for(int i=0; i<n; i++)
            endq[i].clear();
        for(int q=1; q<pc; q++)
            endq[middle[q]-radius[q]].push_back(q);
        for(int i=0; i<n; i++)
            events[i].clear();
        for(int i=1; i<n; i++)
            if (pal[i]>0)
                events[i-pal[i]].push_back(i);
        set<int> S;
        for(int i=0; i<n; i++)
        {
            if (S.find(i)!=S.end())
                S.erase(i);
            for(int e : events[i])
                S.insert(e);
            for(int q : endq[i])
            {
                shorter[q] = query(i,S,middle[q]-1);
                half[q] = query(i,S,middle[q]-radius[q]+radius[q]/2);
            }
        }
        dp[0] = 0;
        dph[0] = 0;
        for(int q = 1; q < pc; q++)
        {
            dph[q] = dph[link[q][0]]+1;
            dph[q] = min(dph[q],dp[half[q]]+radius[q]-2*radius[half[q]]);

            dp[q] = dp[link[q][0]]+2;
            dp[q] = min(dp[q],dph[q]+1);
            dp[q]           =           min(dp[q],dp[shorter[q]]+2*radius[q]-
2*radius[shorter[q]]);
        }

        int best = n;
        for(int q = 0; q < pc; q++)
            best = min(best, n - 2*radius[q] + dp[q]);
        cout << best << endl;
    }
}
```

CÁCH 2 : Sử dụng cấu trúc dữ liệu Palindrome Tree

```
#include <bits/stdc++.h>
#define CHECK(cond) do { if (!(cond)) {\
  exit(1);\
} } while(0)
using namespace std;
```

```
struct node {
  // Tiền tố thích hợp dài nhất của palindrome cũng là hậu tố của nó.
  node *prefsuf;
  // Tiền tố thích hợp dài nhất của hậu tố palindrome và ít nhất gấp đôi
  node *prefsuf2;
  // Palindrome kết quả từ việc loại bỏ các ký tự đầu tiên và cuối cùng.
  node *shorter;
  // Độ dài thực tế của prefix-suffix, từ prefsuf có thể khởi tạo điểm  ban
đầu trỏ đến //Palindrome dài hơn mà cần được rút ngắn bằng các liên kết
« ngắn hơn »
  int prefsuf_len;
  // Vị trí ở giữa và độ dài của palindrome.
  int p, len;
  // Số lượng hoạt động tối đa chúng ta có thể lưu xây dựng palindrome này.
  int saved;
  // Danh sách các con tạm thời, DFS trên cây.
  list<node*> children;
  // Danh sách các nút trỏ đến nút này với prefsuf.
  list<node*> prefsuf_targets;
  node(node *shorter_, int p_, int len_)
      : prefsuf(0), prefsuf2(0),
        shorter(shorter_), prefsuf_len(0), p(p_), len(len_), saved(-1) {}
  int end() { return p + len; }
  int start() { return p - len; }
};
struct graph {
  // Với mỗi vị trí trong xâu con trỏ tới nút đại diện palindrome dài nhất
tập trung tại nút này.
  vector<node *> L;
  // Danh sách các con trỏ tới tất cả các nút.
  vector<node *> all;
  // Bán kính của palindrome lớn nhất tập trung tại kí tự được.
  vector<int> radius;
  void build(const char *w, int n) {
    vector<node *> G;
    L.clear();
    L.resize(n+1, 0);
    radius.clear();
    radius.resize(n+1, 0);
    all.push_back(new node(0, 0, 0));
    L[0] = all[0];
    node *prev = all[0];
    for(int i = 0; i < n;) {
      G.clear();
      while (0 <= i - radius[i] - 1 && i + radius[i] < n &&
             w[i + radius[i]] == w[i - radius[i] - 1]) {
        prev = new node(prev, i, ++(radius[i]));
        all.push_back(prev);
        G.push_back(prev);
      }
      L[i] = prev;
      prev = all[0];
      int g = i, G_start = 0;
      radius[++i] = 0;
      if (G.empty()) for (; i < radius[g] + g; ++i) {
        node *n = L[g];
        CHECK(2*g-i>=0);
        CHECK(n);
        int R = g + radius[g] - i, r = radius[2*g-i];
        if (R > r) {
          radius[i] = r;
          node *m = L[2*g-i];
          L[i] = m;
```

```
          radius[i+1] = 0;
        } else {
          prev = L[2*g-i];
          CHECK(prev->len == r);
          radius[i] = R;
          for (int j = 0; j < r-R; ++j)
            prev = prev->shorter;
          L[i] = prev;
          goto next;
        }
      } else for (; i < radius[g] + g; ++i) {
        CHECK(G_start < G.size());
        while(true) {
          node *n = G[G_start];
          CHECK(2*g-i>=0);
          CHECK(n);
          int R = n->end() - i, r = radius[2*g-i];
          if (R > r) {
            radius[i] = r;
            node *m = L[2*g-i];
            L[i] = m;
            radius[i+1] = 0;
            break;
          } else {
            G_start++;
            node *m = L[2*g-i];
            CHECK(m->len == r);
            n->prefsuf = m;
            n->prefsuf_len = R;
            if (G_start == G.size()) {
              radius[i] = R;
              prev = m;
              for (int j = 0; j < r-R; ++j)
                prev = prev->shorter;
              L[i] = prev;
              goto next;
            }
          }
        }
      }
      next:;
    }
  }
  ~graph() {
    for (int i = 0; i < all.size(); ++i) delete all[i];
  }
  void adjust_prefsuf();
  void adjust_prefsuf2();
  void make_prefsuf_targets();
  void make_prefsuf_children();
  int dp();
};
void graph::make_prefsuf_targets() {
  for (int i = 0; i < all.size(); ++i) {
    all[i]->children.clear();
    all[i]->prefsuf_targets.clear();
  }
  for (int i = 0; i < all.size(); ++i) {
    node *n = all[i];
    if (n->shorter)
      n->shorter->children.push_back(n);
    if (n->prefsuf)
      n->prefsuf->prefsuf_targets.push_back(n);
```

```
  }
}
void adjust_prefsuf_rec(node *n, vector<node *> *stack) {
  stack->push_back(n);
  for (list<node*>::iterator it = n->prefsuf_targets.begin();
                             it != n->prefsuf_targets.end(); ++it) {
    node *m = *it;
    CHECK(m->prefsuf_len < stack->size());
    m->prefsuf = (*stack)[m->prefsuf_len];
  }
  for (list<node*>::iterator it = n->children.begin(); it != n-
>children.end();
       ++it) {
    adjust_prefsuf_rec(*it, stack);
  }
  stack->pop_back();
}
// Điều chỉnh các con trỏ prefsuf để chúng trỏ sang palindrome bên phải
(ngắn).
void graph::adjust_prefsuf() {
  make_prefsuf_targets();
  for (int i = 0; i < all.size(); ++i) {
    node *n = all[i];
    if (n->shorter) continue;
    vector<node *> stack;
    adjust_prefsuf_rec(n, &stack);
  }
void graph::make_prefsuf_children() {
  for (int i = 0; i < all.size(); ++i) {
    all[i]->children.clear();
  }
  for (int i = 0; i < all.size(); ++i) {
    node *n = all[i];
    if (!(n->prefsuf)) continue;
    n->prefsuf->children.push_back(n);
  }
}
bool prefsuf2_compare(const node *a, const node *b) {
  return a->len < b->len;
}
void adjust_prefsuf2_rec(node *n, vector<node *> *stack) {
  if (!stack->empty()) {
    CHECK(stack->back()->len < n->len);
  }
  node fake_node(0,0,0);
  fake_node.len = n->len/2;
  vector<node*>::iterator it = upper_bound(
      stack->begin(), stack->end(), &fake_node, prefsuf2_compare);
  if (it != stack->begin()) {
    CHECK(it == stack->end() || (*it)->len > n->len/2);
    --it;
    n->prefsuf2 = *it;
    CHECK(n->prefsuf2->len <= n->len/2);
  } else {
    n->prefsuf2 = 0;
  }
  stack->push_back(n);
  for (list<node*>::iterator it = n->children.begin(); it != n-
>children.end();
       ++it) {
    adjust_prefsuf2_rec(*it, stack);
  }
  stack->pop_back();
```

```
}
// Tìm prefix-suffixes phù hợp với half-palindromes.
void graph::adjust_prefsuf2() {
  make_prefsuf_children();
  for (int i = 0; i < all.size(); ++i) {
    node *n = all[i];
    if (n->prefsuf) continue;    vector<node *> stack;
    adjust_prefsuf2_rec(n, &stack);
  }
}

int dp_rec(node *n) {
  if (n->saved != -1) return n->saved;
  int val = n->len > 0 ? n->len - 1 : 0;
  if (n->prefsuf2)
    val = dp_rec(n->prefsuf2) + n->len - 1;
  if (n->shorter)
    val = max(val, dp_rec(n->shorter)+(n->len>1));
  return n->saved = val;
}
// Tính kết quả thực tế.
int graph::dp()
{
  int saved = 0;
  for (int i = 0; i < all.size(); ++i)
    saved = max(saved, dp_rec(all[i]));
  return saved;
}
int alg(const char *w, int n) {
  std::vector<node*> nodes(n);
  graph G;  G.build(w, n);
  G.adjust_prefsuf();
  G.adjust_prefsuf2();
  int result = G.dp();
  return n - result;
}

int main()
{
  int Z;
  freopen("VIRUS.INP", "r", stdin);
   freopen("VIRUS.OUT", "w", stdout);
  cin >> Z;
  while(Z--) {
    string s;    cin >> s;
    cout << alg(s.c_str(), s.size()) << endl;
  }
}
```

❖ **Nhận xét độ phức tạp:** Chủ yếu thời gian của chương trình phụ thuộc vào thời gian tính tính chi phí tối thiểu để tổng hợp các xâu Palindrome p_1, p_2, …, p_s theo thứ tự tăng dần của độ dài xâu và việc thêm một kí tự đầu tiên vào một nửa của xâu Palindrome p_i. Tổng hợp xâu Palindrome p_l là tiền tố của một nửa xâu Palindrome p_i và thêm một số chữ cái vào cuối => độ phức tạp là: *O(nlogn)*

ĐỘ PHỨC TẠP: *O(nlogn).*

❖ **Test:**

https://drive.google.com/file/d/1WQ8ZFuQrBdwGSB9jnty2dQy_X4RzvrZy/view?usp=sharing

❖ **Cảm nhận:**

Đây là bài toán khá hay của lớp bài toán Palindrome với độ dài lớn mà áp dụng cấu trúc Palindrome Tree. Ngoài việc phải tìm tất cả các xâu con Palindrome từ xâu ban đầu. Ta phải tính thêm chi phí để tổng hợp các xâu con đó theo độ dài tăng dần. Tại mỗi bước tổng hợp ta thêm vào kí tự đầu tiên là **tổng tiền tố**.

II.3.5. Bài toán xâu con Palindrome dài nhất LPS

Bài toán:

Một palindrome là một xâu mà đọc xuôi cũng giống như đọc ngược lại của nó. Ví dụ "malayalam", "dad", "appa", v.v ...

Yêu cầu: Tìm độ dài của xâu con tiếp giáp dài nhất của một xâu đã cho mà là một Palindrome.

Dữ liệu: Vào từ file văn bản **LPS.INP** gồm:

- Dòng thứ nhất: chứa một số nguyên dương N duy nhất là số lượng kí tự trong xâu.
- Dòng thứ hai: là một xâu S có N kí tự, trong đó các kí tự luôn là chữ cái tiếng Anh viết thường, tức là từ 'a' đến 'z'.

Kết quả: Ghi ra file văn bản **LPS.OUT** gồm một dòng duy nhất ghi một số nguyên biểu thị độ dài của xâu con palindrome dài nhất.

LPS.INP	LPS.OUT
5 ababa	5
16 forgeeksskeegfor	10

Ràng buộc:

- ✓ $1 \leq N \leq 10^5$
- ✓ Xâu S chỉ bao gồm các chữ cái latinh in thường.
- ✓ Sub1: 30% test có $1 \leq N \leq 100$.
- ✓ Sub2: 40% test có $N \leq 1000$.
- ✓ Sub3: 30% test có $N \leq 10^5$.

❖ **Xác định bài toán:**

Input: Số nguyên dương N ($1 \leq N \leq 10^5$); xâu S gồm N kí tự chữ cái thường

Output: Độ dài xâu con Palindrome dài nhất.

❖ **Phân tích thuật toán:**

CÁCH 1: Sử dụng thuật toán Manacher với độ phức tạp *O(n).*

=> Độ phức tạp tuyến tính.

Cài đặt chương trình

```
#include <bits/stdc++.h>
#include <ext/algorithm>
#include <ext/numeric>
using namespace std;
using namespace __gnu_cxx;
#define endl '\n'
vector<int> manacher(const string &s)
{
   int n = 2 * s.length();
   vector<int> rad(n);
   for (int i = 0, j = 0, k; i < n; i += k, j = max(j - k, 0))
   {
         for( ; i >= j && i + j + 1 < n
               && s[(i - j) / 2] == s[(i + j + 1) / 2]; ++j);
         rad[i] = j;
         for (k = 1; i >= k && rad[i] >= k
               && rad[i - k] != rad[i] - k; ++k)
               rad[i + k] = min(rad[i - k], rad[i] - k);
   }
   return rad;
}
int main()
{
    freopen("LPS.INP","r",stdin);
    freopen("LPS.OUT","w",stdout);
   ios_base::sync_with_stdio(0);
   cin.tie(0);
   int n;
   string s;
   cin >> n >> s;
   auto rad = manacher(s);
   cout << *max_element(rad.begin(), rad.end()) << endl;
   return 0;
}
```

CÁCH 2: Sử dụng Palindrome Tree

Đây là bài toán có thể ứng dụng cấu trúc dữ liệu cây Palindrome. Cấu trúc cây Palindrome gần với một đồ thị có hướng. Nó là sự hợp nhất của hai câycó chung một số nút chung. Mỗi nút lưu trữ một xâu con Palindrome của xâu đã cho bằng cách lưu trữ chỉ số của chúng.

Cây này gồm có hai loại cạnh:

- ✓ Cạnh chèn: cạnh có trọng số
- ✓ Palindrome hậu tố lớn nhất: không có trọng số.

Cạnh chèn:

Chèn từ một nút u đến v với trọng số x có nghĩa là nút v được hình thành bằng cách chèn x ở đầu và cuối của xâu tại u. Vì 'u' đã là một palindrome, do đó xâu kết quả tại nút v cũng sẽ là một palindrome. x sẽ là một kí tự duy nhất cho mọi cạnh. Do đó, một nút có thể có tối đa 26 cạnh chèn (xem xét xâu kí tự ngắn hơn).

Cạnh Palindrome hậu tố lớn nhất:

Với mỗi một nút cạnh này sẽ chỉ đến nút là xâu Palindrome hậu tố lớn nhất. Để đơn giản ta gọi là cạnh hậu tố => Mọi nút chỉ có một cạnh hậu tố vì ta sẽ không lưu trữ các cạnh trùng lặp trong cây.

Ta sẽ tạo ra tất cả các xâu con palindrome và sau đó trả về cái cuối cùng ta có, vì đó sẽ là xâu con palindrome dài nhất.

Vì cây Palindrome lưu trữ các palindrome theo thứ tự đến của một kí tự nhất định nào đó. Do đó, xâu con Palindrome dài nhất sẽ luôn ở chỉ số cuối cùng của mảng cây.

❖ **Cài đặt chương trình:**

Code 1:

```
#include <bits/stdc++.h>
using namespace std;
 #define MAXN  1000

 struct Node
{
    // Lưu trữ chỉ số đầu và cuối bao gồm cả nút hiện tại
    int start, end;
      // lưu trữ độ dài của xâu con
    int length;
      // lưu trữ nút chèn cho tất cả các kí tự từ 'a' đến 'z'
    int insertionEdge[26];
      // lưu trữ nút Palindrome hậu tố lớn nhất cho nút hiện tại
    int suffixEdge;
};
// Hai nút giả đã giải thích ở trên
Node root1, root2;
  // lưu thông tin của nút mà truy cập thời gian không thay đổi
Node tree[MAXN];
// Theo dõi nút hiện tại trong khi chèn
int currNode;
string s;
int ptr;

  // Hàm chèn cạnh vào trong cây
void insert(int currIndex)
{
    // Tìm X sao cho s[currIndex] + X + s[currIndex] là palindrome.
    int temp = currNode;
      while (true)
    {
        int currLength = tree[temp].length;
        if (currIndex - currLength >= 1 &&
            (s[currIndex] == s[currIndex -  currLength - 1]))
              break;
          temp = tree[temp].suffixEdge;
    }
      // Kiểm tra nếu s[currIndex] + X + s[currIndex] là trong cây hiện tại
rồi.
    if (tree[temp].insertionEdge[s[currIndex] -  'a'] != 0)
    {
        currNode = tree[temp].insertionEdge[s[currIndex] - 'a'];
        return;
    }
    // còn không tạo nút mới;
    ptr++;
      tree[temp].insertionEdge[s[currIndex] - 'a'] = ptr;
      tree[ptr].end = currIndex;
```

```
    tree[ptr].length = tree[temp].length + 2;
    tree[ptr].start = tree[ptr].end - tree[ptr].length + 1;
    // Đặt cạnh hậu tố cho Nút mới được tạo.
    currNode = ptr;
    temp = tree[temp].suffixEdge;
    // Palindrome hậu tố dài nhất cho xâu có độ dài 1 là rỗng
    if (tree[currNode].length == 1) {
        tree[currNode].suffixEdge = 2;
        return;
    }
    // Else
    while (true)
{
        int currLength = tree[temp].length;
        if (currIndex - currLength >= 1 &&
            (s[currIndex] ==
             s[currIndex - currLength - 1]))
               break;
        temp = tree[temp].suffixEdge;
    }
    tree[currNode].suffixEdge =
        tree[temp].insertionEdge[s[currIndex] - 'a'];
}

// Driver code
int main()
{
  freopen("LPS.INP", "r", stdin);
   freopen("LPS.OUT", "w", stdout);
    // Các điểm đỉnh hậu tố của gốc ảo trỏ đến chính nó, từ đó một xâu ảo có
độ dài =-1 có xâu hậu tố ảo và gốc ảo
    root1.length = -1;
    root1.suffixEdge = 1;
    //các điểm đỉnh hậu tố của gốc rỗng trỏ tới gốc ảo, từ đó một xâu có độ
dài = 0 có một xâu hậu tố ảo.
    root2.length = 0;
    root2.suffixEdge = 1;
    tree[1] = root1;
    tree[2] = root2;
    ptr = 2;
    currNode = 1;
    cin>>s;
    for (int i = 0; i < s.size(); i++)
          insert(i);
    // cuối cùng sẽ là chỉ số của xâu con cuối cùng cần tìm
    int last = ptr;
    for (int i = tree[last].start;
         i <= tree[last].end; i++)
               cout << s[i].length;
    return 0;
}
```

Code 2:

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = 1e5 + 1;

int s[maxn], to[maxn][26], len[maxn], link[maxn];
int n, sz, last;

void init()
{
```

```
    s[n++] = -1;
    link[0] = 1;
    len[1] = -1;
    sz = 2;
}

int get_link(int v)
{
    while(s[n - len[v] - 2] != s[n - 1]) v = link[v];
    return v;
}

int add_letter(char c)
{
    s[n++] = c -= 'a';
    last = get_link(last);
    if(!to[last][c])
    {
        len[sz] = len[last] + 2;
        link[sz] = to[get_link(link[last])][c];
        to[last][c] = sz++;
    }
    last = to[last][c];
    return len[last];
}

int main()
{
    freopen("LPS.INP","r",stdin);
    freopen("LPS.OUT","w",stdout);
    ios::sync_with_stdio(0);
    cin.tie(0);
    init();    int n;
    cin >> n;    string s;    cin >> s;
    int ans = 0;
    for(auto c: s)
        ans = max(ans, add_letter(c));
    cout << ans;
    return 0;
}
```

- **Nhận xét độ phức tạp:** Khi sử dụng cấu trúc dữ liệu cây Palindrome bài toán sẽ có thuật toán tối ưu nhất .

  Độ phức tạp thuật toán phụ thuộc vào thuật toán xây dựng cây Palindrome, tại mỗi bước xây dựng ta phải chèn thêm vào một nút là Palindrome mới

  ⇨ **Độ phức tạp :** *O(nlogn)*

- **Test :**

https://drive.google.com/file/d/1Ki2Q5EJ1X4lquRzaHnV0IQrLpOqKcSlH/view?usp=sharing

II.3.6. Bài toán số Palindrome NUMOFPAL

(Nguồn http://www.spoj.com/problems/NUMOFPAL)

Bài toán:

Mỗi palindrome có thể luôn được tạo từ các palindrome khác, một ký tự đơn cũng là một palindrome. Ví dụ: xâu "malayalam" có thể được tạo bằng một số cách:

- ✓ malayalam = m + ala + y + ala + m
- ✓ malayalam = m + a + l + aya + l + a + m

Giá trị của hàm NumPal (*s*) là số lượng các palindrome khác nhau có thể được tạo bằng xâu *s* theo phương pháp trên. Nếu cùng một palindrome xuất hiện nhiều hơn một lần thì tất cả chúng đều được tính riêng.

Yêu cầu: Cho trước xâu *s* hãy tính giá trị của hàm NumPal(*s*).

Dữ liệu: Vào từ file văn bản **NUMOFPAL.INP** gồm duy nhất một dòng chứa xâu *s* có độ dài xâu không quá 10^3.

Kết quả: Ghi ra file văn bản **NUMOFPAL.OUT** gồm một dòng duy nhất ghi một số nguyên là kết quả của hàm NumPal (*s*).

NUMOFPAL.INP	NUMOFPAL.OUT
malayalam	15

Giới hạn: $1 \le |s| \le 10^3$.

❖ **Xác định bài toán:**

Input: xâu *s* gồm tối đa 10^3 kí tự.

Output: Kết quả của hàm NumPal(*s*).

❖ **Hướng dẫn thuật toán:**

Đây là một dạng bài toán liên quan đến xâu Palindrome. Một xâu palindrome được kết hợp lại từ các xâu Palindrome con liên tiếp.

Có nhiều thuật toán để giải quyết bài toán này:

- ➢ Sử dụng duyệt thuần túy kết hợp với kiểm tra một xâu có là Palindrome hay không => độ phức tạp $O(n^3)$.
- ➢ Sử dụng thuật toán Manacher với độ phức tạp $O(n)$ kết hợp với tổng tiền tố
- ➢ Sử dụng thuật toán cây Palindrome với độ phức tạp $O(n)$.

❖ **Cảm nhận:**

Đây là bài toán phát triển từ bài toán thứ nhất đếm số xâu con liên tiếp là xâu Palindrome.

II.3.7. Bài toán Palindromes và Siêu năng lực

(Nguồn http://acm.timus.ru/problem.aspx?num=1960)

Bài toán:

Sau khi giải quyết bảy vấn đề trên Timus bằng một từ "Palindrome" trong danh sách vấn đề, Misha có một khả năng khác thường. Bây giờ, khi anh ta đọc một từ, anh ta có thể đếm được số lượng các xâu khác rỗng duy nhất của từ này là palindrome.

Dima muốn thử khả năng mới của Misha. Anh ta thêm các chữ cái s_1, ..., s_n vào một từ, từng chữ cái một và sau mỗi chữ cái hỏi Misha, có bao nhiêu từ palindrome khác rỗng

khác nhau có chứa các xâu con. Trong n số Misha sẽ nói, anh ta sẽ không bao giờ là sai lầm?

Yêu cầu: Cho trước xâu s hãy tính giá trị của hàm NumPal(s).

Dữ liệu: Vào từ file văn bản **PALSUPER.INP** gồm duy nhất một dòng chứa xâu $s_1...s_n$, trong đó s_i là một kí tự chữ cái latinh thường, $1 \le n \le 10^5$.

Kết quả: Ghi ra file văn bản **PALSUPER.OUT** gồm một dòng ghi n số cách nhau bởi dấu cách, số thứ i là số lượng xâu khác rỗng khác nhau của tiền tố s_1 ... s_i là các palindrome.

Ví dụ:

PALSUPER.INP	PALSUPER.OUT
aba	1 2 3

Giới hạn: $1 \le n \le 10^5$.

- Thời gian: 1.0 giây
- Bộ nhớ: 64 MB

❖ **Xác định bài toán:**

Input: xâu $s_1s_2...s_n$ với $n \le 10^5$.

Output: n số nguyên, số thứ i là số lượng xâu khác rỗng khác nhau của tiền tố s_1 ... s_i là các palindrome.

❖ **Hướng dẫn thuật toán:**

Đây là một dạng bài toán liên quan đến xâu Palindrome. Một xâu palindrome được kết hợp lại từ các xâu Palindrome con liên tiếp. Với mỗi xâu con s_1 ... s_i thì ta xác định i là số lượng xâu khác rỗng khác nhau trong xâu con này.

❖ **Cảm nhận:**

Đây là bài toán ứng dụng Palindrome Tree hiệu quả, phát triển từ bài toán thứ nhất đếm số xâu con liên tiếp là xâu Palindrome.

II.3.8. Bài toán 31 Palindromes

(Nguồn http://acm.timus.ru/problem.aspx?space=1&num=2044)

Bài toán:

Đối với mọi tiền tố của xâu đã cho, hãy xác định xem có thể chia thành 1, 2, 3, 4, 5, … 31 xâu Palindrome khác rỗng hay không?

Dữ liệu: Vào từ file văn bản **31_PAL.INP** gồm duy nhất một dòng chứa xâu s gồm n kí tự chữ cái latinh thường, $1 \le n \le 3.10^5$.

Kết quả: Ghi ra file văn bản **31_PAL.OUT** gồm n dòng, mỗi dòng ghi một số nguyên dương, dòng thứ i chứa một số thập phân. Nếu bạn xem xét biểu diễn nhị phân của số thập phân này thì chữ số của nó ở vị trí (j - 1) phải bằng một nếu tiền tố của độ dài i có thể được chia thành j palindrome và khác 0.

31_PAL.INP	31_PAL.OUT
abaa	1 2 5 14

Giới hạn: $1 \le n \le 3.10^5$.

- Thời gian: 0.5 giây
- Bộ nhớ: 64 MB

❖ **Xác định bài toán:**

Input: xâu *s* gồm *n* kí tự, với $1 \le n \le 3.10^5$.

Output: *n* số nguyên dương, số thứ *i* là số thập phân mà biểu diễn nhị phân của nó ở vị trí (*j* - 1) phải bằng một nếu tiền tố của độ dài *i* có thể được chia thành *j* palindrome và khác 0.

❖ **Hướng dẫn thuật toán:**

Đây là một dạng bài toán liên quan đến xâu Palindrome.

Chẳng hạn xét xâu ban đầu *s* =’abaa’ ta thấy xâu *s* có độ dài *n* = 4.

Xâu abaa có thể có các cách chia thành các palindrome con khác nhau và khác 0 như sau:

- ✓ abaa= aba|a
- ✓ abaa= a|b|aa
- ✓ abaa= a|b|a|a

Sử dụng cấu trúc dữ liệu cây Palindrome ta biết, xâu có độ dài bằng một đều là palindrome. Khi đó ta có các Palindrome là các nút *a, b, a, a.* Ngoài ra xâu có cùng một loại kí tự cũng là Palindrome, thêm cạnh từ *a* đến *b* ta có nút *aba* là Palindrome theo cách xây dựng cây palindrome.

Sử dụng đổi một số từ hệ thập phân sang nhị phân ta có:

$1_{10} = 1_2$; $2_{10} = 10_2$; $5_{10} = 101_2$; $14_{10} = 1110_2$;

II.3.9. Bài toán sự phong phú của từ

(Nguồn http://acm.timus.ru/problem.aspx?num=2045)

Bài toán:

Đối với mỗi số nguyên *i* từ 1 đến *n* , bạn phải in một xâu s_i có độ dài *n* bao gồm các chữ cái Latinhh viết thường. Xâu s_i phải chứa chính xác *i* xâu con palindrome khác nhau. Hai xâu con được coi là khác nhau nếu chúng là các xâu khác nhau.

Yêu cầu: Cho trước số nguyên dương *n*, hãy in ra các xâu s_i có thể thỏa mãn hoặc không.

Dữ liệu: Vào từ file văn bản **RICHW.INP** gồm duy nhất số nguyên dương *n*.

Kết quả: Ghi ra file văn bản **RICHW.OUT** gồm n dòng. Nếu đối với một số i , câu trả lời tồn tại, hãy in nó dưới dạng: "$i : s_i$", trong đó s_i là một trong những xâu có thể. Nếu không hãy in ra "i : NO".

RICHW.INP	RICHW.OUT
4	1 : NO 2 : NO 3 : abca 4 : bbca

Giới hạn: $1 \le n \le 2.10^3$.

- Thời gian: 0.5 giây
- Bộ nhớ: 64 MB

❖ **Xác định bài toán:**

Input: số nguyên dương $1 \le n \le 2.10^3$.

Output: đối với mỗi số nguyên i ($1 \le i \le n$) kiểm tra xem câu trả lời có tồn tại hay không và in kết quả tương ứng theo mẫu..

❖ **Hướng dẫn thuật toán:**

Đây là một dạng bài toán liên quan đến xâu Palindrome.

Nhận xét:

- Với n = 1 thì không tồn tại xâu con s_i nào mà chứa chính xác i xâu con palindrome khác nhau.
- Với n = 2 thì không tồn tại xâu con s_i nào mà chứa chính xác i xâu con palindrome khác nhau.
- Với n ≥ 3 ta sử dụng cấu trúc cây Palindrome xuất phát từ mỗi kí tự là một nút chứa Palindrome có độ dài 1.
- Tại mỗi bước ta xây dựng thêm cạnh để tạo thành một Palindrome mới. Nếu xâu con mới tạo thành thỏa mãn thì ta ghi nhận và thoát khỏi vòng lặp.

II.3.10. Bài toán sự phong phú của các từ nhị phân

(Nguồn https://acm.timus.ru/problem.aspx?space=1&num=2037)

Bài toán:

Đối với mỗi số nguyên i từ 1 đến n , bạn phải in một xâu s_i có độ dài n chỉ bao gồm các chữ cái a và b. Xâu s_i phải chứa chính xác i xâu con palindrome khác nhau. Hai xâu con được coi là khác nhau nếu chúng là các xâu khác nhau.

Yêu cầu: Cho trước số nguyên dương n, hãy in ra các xâu s_i có thể thỏa mãn hoặc không.

Dữ liệu: Vào từ file văn bản **RICHBW.INP** gồm duy nhất số nguyên dương n.

Kết quả: Ghi ra file văn bản **RICHBW.OUT** gồm n dòng. Nếu đối với một số i , câu trả lời tồn tại, hãy in nó dưới dạng: "$i : s_i$", trong đó s_i là một trong những xâu có thể. Nếu không hãy in ra "i : NO".

RICHBW.INP	RICHBW.OUT
4	1 : NO 2 : NO 3 : NO 4 : aaaa

Giới hạn: $1 \le n \le 2.10^3$.

- Thời gian: 0.5 giây
- Bộ nhớ: 64 MB

❖ **Xác định bài toán:**

Input: số nguyên dương $1 \le n \le 2.10^3$.

Output: đối với mỗi số nguyên i ($1 \le i \le n$) kiểm tra xem câu trả lời có tồn tại hay không và in kết quả tương ứng theo mẫu..

❖ **Hướng dẫn thuật toán:**

Đây là một dạng bài toán liên quan đến xâu Palindrome.

Nhận xét:

- Với n = 1 hoặc n = 2 hoặc n = 3 thì không tồn tại xâu con s_i nào mà chứa chính xác i xâu con palindrome khác nhau.
- Với n ≥ 4 ta sử dụng cấu trúc cây Palindrome xuất phát từ 2 kí tự '*a'* và kí tự 'b' là hai Palindrome có độ dài 1 => ta xây dựng cây có 2 nút ban đầu là *a* và *b.*
- Tiến hành xây dựng cây Palindrome bằng cách tại mỗi bước ta xây dựng thêm cạnh để tạo thành một Palindrome mới. Nếu xâu con mới tạo thành thỏa mãn thì ta ghi nhận và thoát khỏi vòng lặp. Vòng lặp bắt đầu từ nút Palindrome *'a'*.

II.3.11. Bài toán máy phát Palindrome

(Nguồn https://www.e-olymp.com/en/problems/2468)

Bài toán:

Xét một xâu g bất kỳ. Ta gọi xâu này *là một máy phát palindrom* . Tập hợp các palindromes *P(g)* được tạo bởi xâu này được định nghĩa như sau.

Đặt độ dài xâu là n . Đối với tất cả i từ **1** đến n, các xâu $g[1..i]g[1..i]^r$ và $g[1..i]$ $g[1..i - 1]^r$ được bao gồm trong *P(g)*, trong đó α^r có nghĩa là α được viết theo thứ tự ngược lại.

Ví dụ, nếu **g** = " **olymp** ", thì **P(g)** = {"**oo**", "**o**", "**ollo**", "**olo**", "**olyylo**", "**olylo**", "**olymmylo**", "**olymylo**", "**olymppmylo**","**olympmylo**"}.

Yêu cầu: Đối với một bộ tạo palindrome *g* và xâu *s* đã cho**,** cần phải tìm số lần xuất hiện của các xâu từ *P(g)* trong *s* làm xâu con. Cụ thể, cần phải tìm số lượng cặp *(i,j)* sao cho *s*[*i..j*] ϵ *P(g)*.

Dữ liệu: Vào từ file văn bản **GENPAL.INP** gồm:

- Dòng đầu tiên: chứa xâu *g*.
- Dòng thứ hai: chứa xâu *s* .

Kết quả: Ghi ra file văn bản **GENPAL.OUT** gồm duy nhất một dòng ghi một số nguyên dương là số lượng cặp *(i,j)* sao cho *s[i..j]* **ϵ** *P(g)*.

Ví dụ:

GENPAL.INP	GENPAL.OUT
olymp olleolleolympmyolylomylo	7

Giới hạn: Độ dài của xâu *s* và $g \leq 10^5$.

- Thời gian: 1.0 giây; Bộ nhớ: 64 MB

❖ **Xác định bài toán:**

Input: Hai xâu *g* và *s* khác rỗng và có tối đa 10^5 kí tự.

Output: số lượng cặp *(i,j)* sao cho *s[i..j]* **ϵ** *P(g)*.

❖ **Hướng dẫn thuật toán:**

Đây là một dạng bài toán liên quan đến xâu Palindrome.

Sử dụng thuật toán cây Palindrome đếm số lượng Palindrome con có thể được tạo thành từ xâu s theo nguyên tắc với mọi *i* từ **1** đến *n*, các xâu con Palindrome *s*[1..*i*]*s*[1..*i*]^r và *s*[1..*i*] *s*[1..*i* - 1]^r trong đó *s*^{r} là xâu ngược của xâu *s* (tham khảo thuật toán và cài đặt chương trình của bài toán II.3.2).

Duyệt trong xâu g, đếm số lượng cặp *(i,j)* sao cho *s[i..j]* **ϵ** *P(g)*.

III. KẾT LUẬN

Sau khi áp dụng sáng kiến vào thực tế dạy học sinh lớp chuyên Tin đặc biệt là các em học sinh trong đội tuyển dự thi chọn học sinh giỏi Tỉnh và Quốc gia, tôi thấy nó mang lại hiệu quả rất rõ rệt. Nó thay đổi cách tiếp cận, phương pháp làm các dạng bài toán liên quan đến xâu Palindrome có kích thước lớn, bây giờ chúng ta đã có thêm một phương pháp giúp cải tiến thuật toán một cách tối ưu nhất để thực hiện. Hầu hết giáo viên và học sinh đều đánh giá đây là một phương pháp hay, cần phổ biến rộng và củng cố, luyện tập để nâng cao được chất lượng giảng dạy, học tập và thi cử của học sinh trong các kì thi có các bài toán liên quan đến xâu Palindrome.

Với thời gian nghiên cứu có hạn, chuyên đề này chắc chắn không tránh khỏi những khiếm khuyết. Tôi xin chân thành cảm ơn những nhận xét, đánh giá và góp ý của các đồng nghiệp để tôi bổ sung và hoàn thiện cho đề tài.

Tôi xin trân trọng cảm ơn !

DANH MỤC TÀI LIỆU THAM KHẢO

1. https://vnoi.info/wiki/translate/codeforces/palindrome-tree
2. https://codeforces.com/blog/entry/13958
3. https://medium.com/@alessiopiergiacomi/eertree-or-palindromic-tree-82453e75025b
4. https://www.geeksforgeeks.org/palindromic-tree-introduction-implementation/
5. http://codeforces.com/
6. https://www.codechef.com/
7. http://codeforces.com/blog/entry/13959
8. http://adilet.org/blog/25-09-14/
9. http://www.spoj.com/problems/NUMOFPAL
10. http://www.spoj.com/problems/LPS
11. http://acm.hdu.edu.cn/showproblem.php?pid=3948
12. https://codeforces.com/gym/100543
13. https://codeforces.com/gym/100548
14. http://codeforces.com/contest/17/problem/E
15. http://acm.timus.ru/problem.aspx?num=1960
16. https://www.e-olymp.com/en/problems/2468