Manacher

一.概念： Manacher算法应用于一个特定场景：查找一个长度为n的字符串S中的最长回文子串(Longest Palindromic Substring)。Manacher算法的复杂度为O(n),是这种场景中效率最高的回文串算法。

二.回文串求解方法

1.中心扩展法

思想：一种比较有效的暴力法是“中心扩展法”。把S的每个字符或每两个相同的字符看作中心，然后左右扩展检查，判断它左右的对称位置是否相同，若相同，则是回文的一部分，直到对称位置不同为止。同时还要考虑奇数长度回文串和偶数长度回文串。

时间复杂度:(a):S中几乎没有长度大于1的回文串。S中的n个字符，共检查$O(n)$次就够了。（b）:S中有大量回文串，且长度较长。时间复杂度：$O（n^2）$

2.Manacher 求回文串

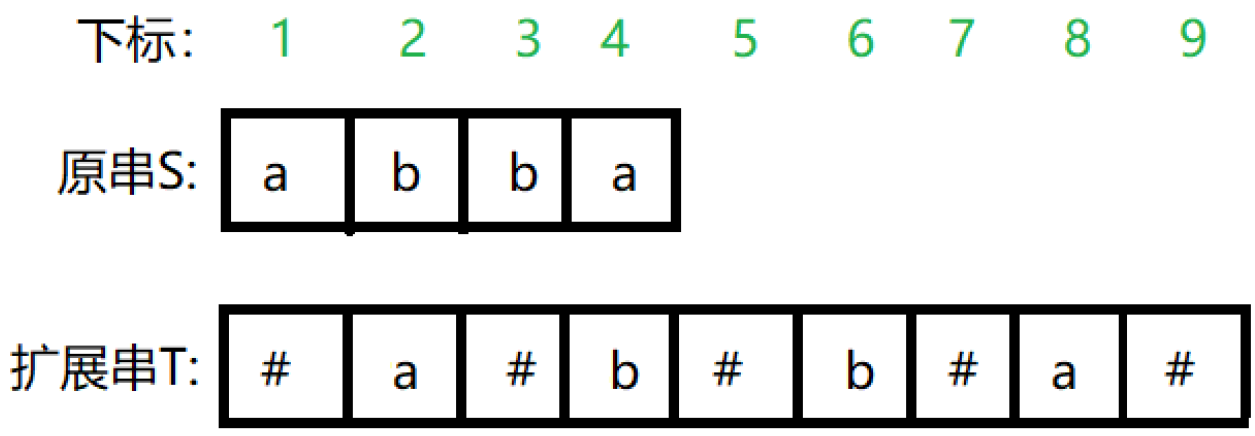
算法步骤：

(1):一些定义

原串：要求解的字符串S

扩展串：在S的每个字符左右插人一个不属于S的字符，如‘#’，记为字符串T。（这样转换后只需要考虑奇数长度的回文串）

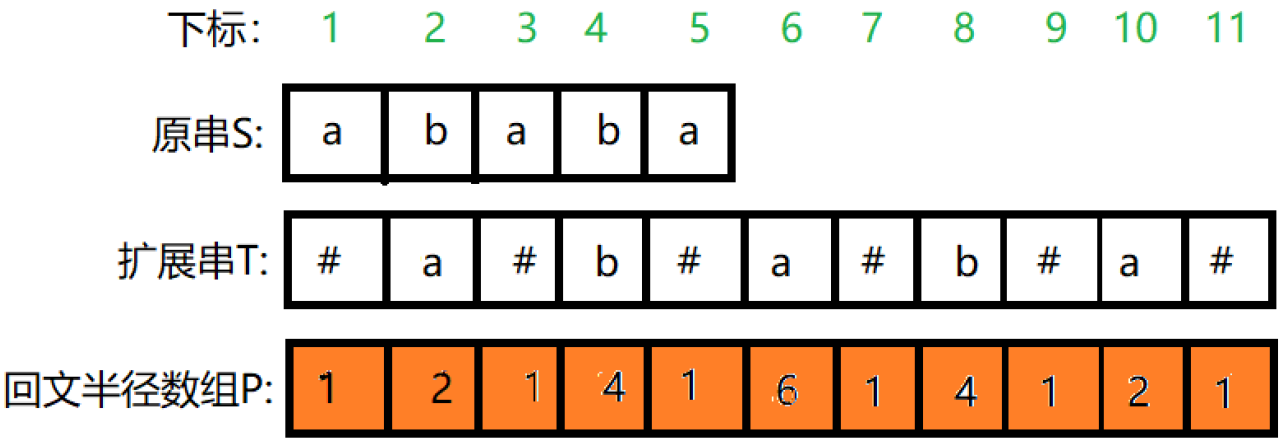
例如下图：



回文中心c：在字符串T中，以字符c作为回文串的中心

回文半径数组P[]：P[i]表示以字符T[i]为回文中心的最长回文的半径。

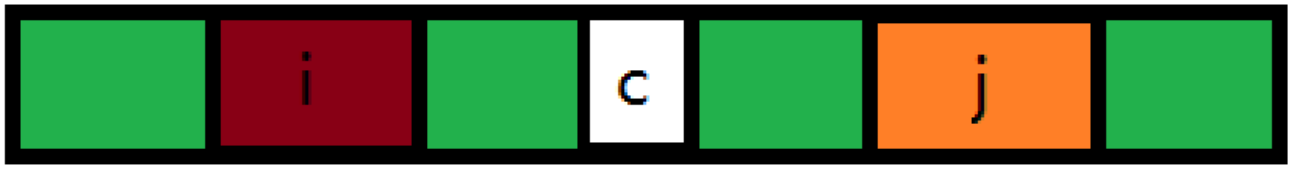
例如下图：



(2):算法核心步骤

基本思想：前面提到中心扩展法低效的原因是有重复的检查，Manacher算法的核心思想是利用“回文的镜像也是回文”这个特征减少重复。

例如下图：



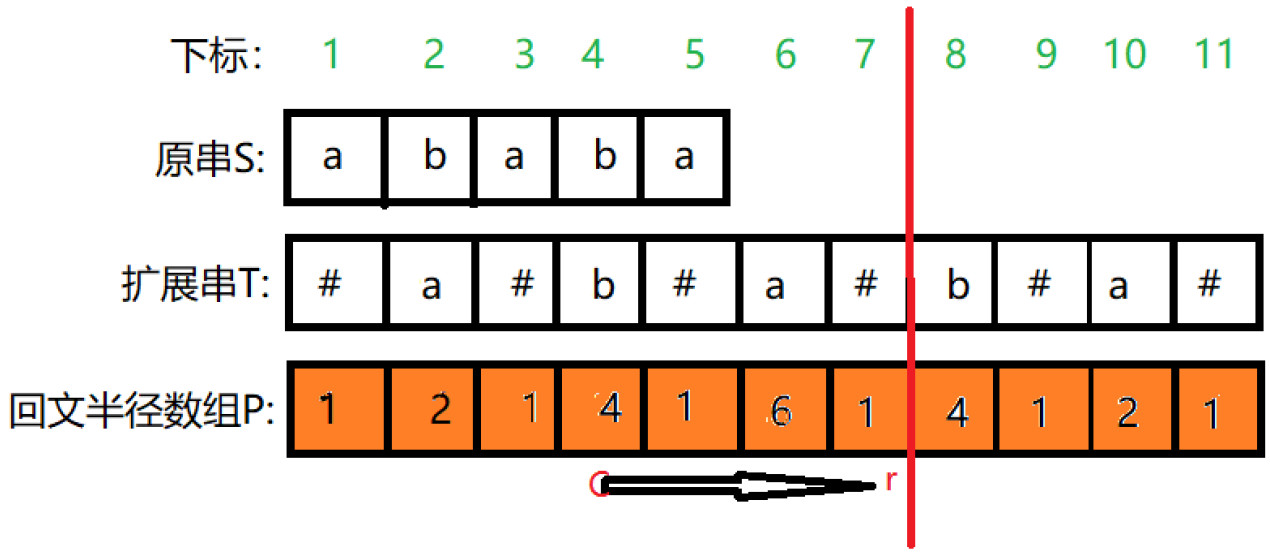
以c为回文中心的回文串长度为整个字符串（绿色部分），那么以j为中心的回文串长度和其关于c的对称点i的回文长度应该相等，那么就可以采用递推的思想，加速求出以每个点为回文中心的回文串长度。

核心步骤：

(a). 从T字符串的下标1到下标n，依次求解回文半径数组，期间记录以字符$T[1-i]$为回文中心情况下回文半径能够向右延伸的最大下标r和其对应的回文中心下标c。

例如下图：

当依次求解$T[1-5]$的回文半径数组时，c=4,r=7;



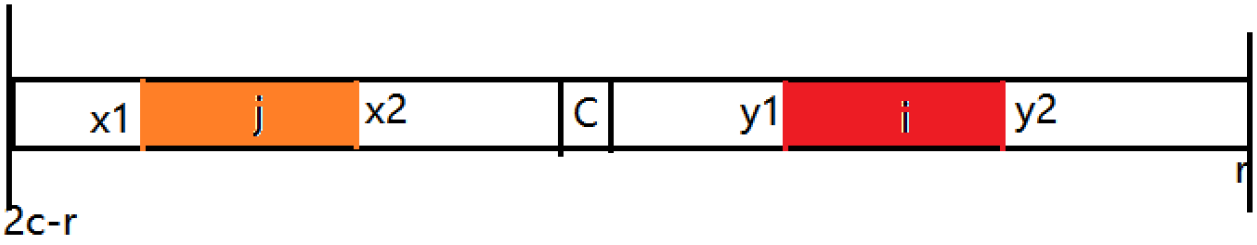
(b) 分情况讨论求解$p[i]$, 当求解到回文中心i时，分如下情况：

情况1：i没被r包住, 此时暴力扩展求解以i位置为回文中心的回文半径，更新r和c

情况2：i被r包住，i关于c的对称点为$2*c-i$, 当对称点$2*c-i$的回文区域在大回文区域内（以c为回文中心的回文串），那么$p[i]=p[2*c-i]$。

例如下图：

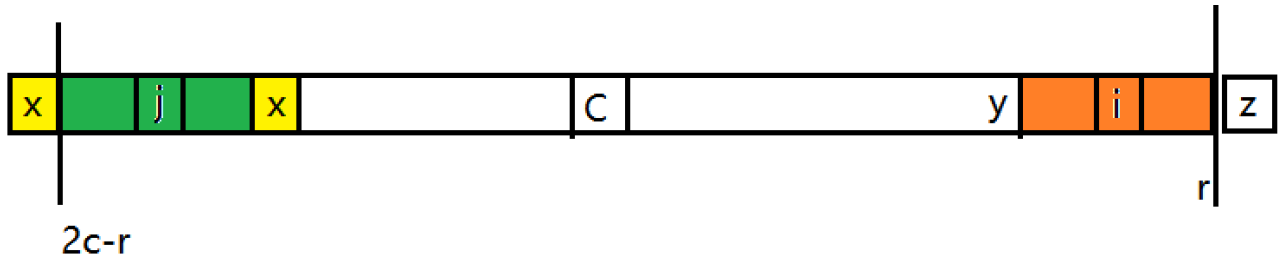
j为i关于c的对称点，所以橙色部分和红色部分相等，所以$p[i]$至少等于$p[j]$, 由于$p[j]$已经确定，所以$x1!=x2, y1$和$x2, y2$和$x1$关于c对称，所以$y1=x2, y2=x1$, 假设$y1==y2$, 那么$x1==x2$, 与之前矛盾，所以$p[i]==p[j]$



情况3：i被r包住，i关于c的对称点为$2*c-i$, 当对称点$2*c-i$的回文区域在大回文区域外，那么$p[i]=r-i+1$

例如下图：

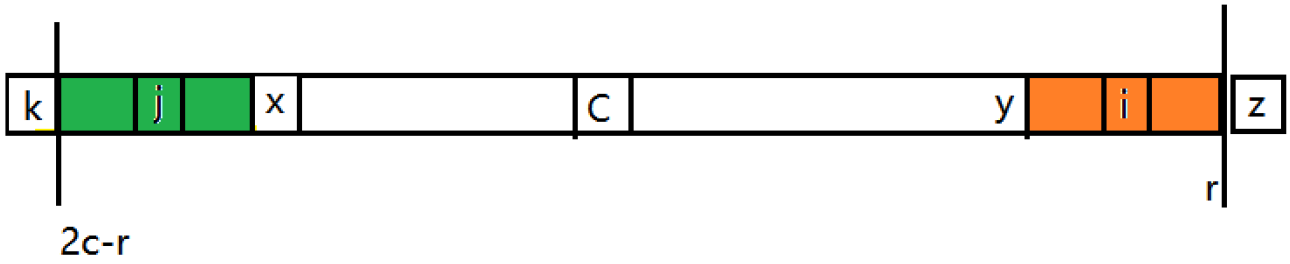
j为i关于c的对称点，所以绿色部分和橙色部分相等，所以$p[i]$至少等于绿色部分（$r-i+1$）, 由于$p[j]$已经确定，y和z必不相等，所以$p[i]=r-i+1$。



情况4：i被r包住，i关于c的对称点为$2*c-i$, 当对称点$2*c-i$的回文区域撞线大回文区域的边界，那么$p[i]=r-i+1$之后，从r继续往后暴力匹配。

例如下图：

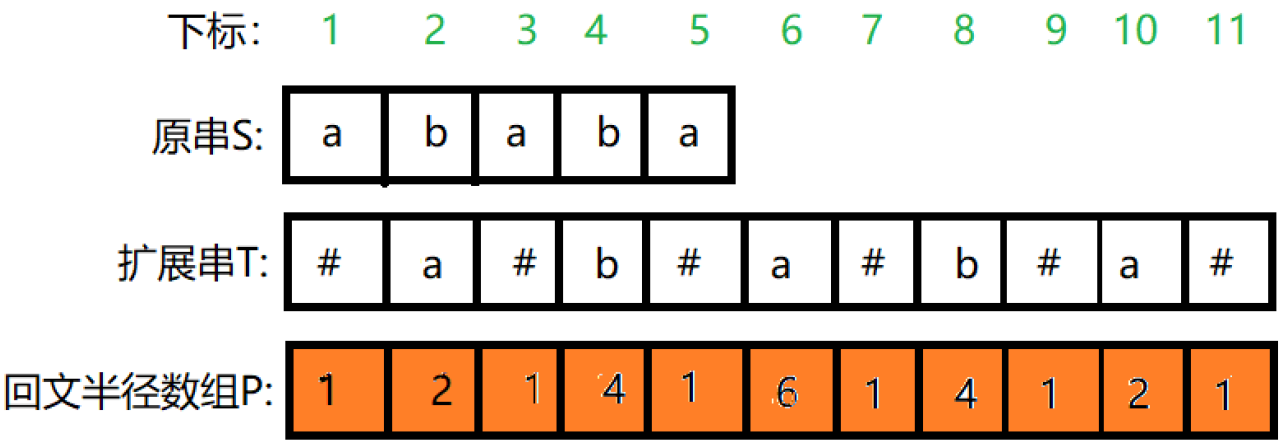
j为i关于c的对称点，所以绿色部分和橙色部分相等，所以$p[i]$至少等于绿色部分（$r-i+1$）, 由对称关系得：$x==y, x!=k, z!=k$, 可以得到，y和z可以相等，也可以不相等，所以$p[i]$需要继续往后匹配。



扩展串与原串的对应关系：

(a). S中的位置i，在T中的位置为$2*i$。

(b). 在T中以位置i为回文中心最大回文半径为$p[i]$，对应到S中，以其为回文中心的整个回文串的长度为$p[i]-1$。



(3):代码实现（洛谷P3805）

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

using namespace std;

const int N = 11000000 + 10;

int p[N << 1], n,m,res;
char a[N], s[N << 1];

void change()
{
    n = strlen(a+1);
    int k = 0;
    s[++k] = '#';
    for (int i = 1; i <= n; i++) {
        s[++k] = a[i];
        s[++k] = '#';
    }
    n = k;

    return;
}
void manacher()
{
    int c = 0, r = 0;
    int len = 0;
    for (int i = 1; i <= n; i++) {
        len = r >= i ? min(p[2 * c - i], r - i) : 1;
        while (i + len <= n && i - len >= 1 && s[i + len] == s[i - len]) len++;
        if (i + len > r) {
            r = i + len;
            c = i;
        }
        res = max(res, len);
        p[i] = len;
    }

    return;
}
int main()
```

```
{
    scanf("%s", a + 1);
    change();
    manacher();
    res = res - 1;
    cout << res << endl;

    return 0;
}
```