### 回文子串的Manacher算法

回文子串可以动态规划递归，速度n\*n

最常规的是迭代每一个字符，看它周围相不相同，直到不相同为止，

为了避免分奇数偶数，在字符串之中插入多余字符字符串

比如对于以下字符串str： daabaac

插入后str是 #d#a#a#b#a#a#c#

设p数组是回文半径，定义单个字符半径是1

看s[0],s[0]左边是边界，那么以为s[0]为中心的回文串半径长度是1

看s[1],,两边看，以为s[1]为中心的回文串半径长度是2 。。。

最后半径序列是：

# d # a # a # b # a # a # c #  
1 2 1 2 1 2 1 6 1 2 5 2 1 2 1

daabaac的最长回文子串长度是这个p数组最大的减去1

这个可以通过关系推出来；

P[i]代表s[i]为中心的回文串半径长度,那么由于这个s的子串每个都是原本字符串str的2\*l+1,那经过计算结果是r=p[i]-1

关键问题是怎么去推p数组，这个常规方法必然是n\*n复杂度

优化速度的方法是，p[i]不从1开始。

对于一个要判断的i下标，提前知道以s[i]为中心的回文半径至少是多少， 怎么提前知道，根据已经扩展出来的半径，记录maxright代表已经扩展出来过的序列最右端，pos代表那个序列的中心

那么分类

如果当前i>maxRight,那么没办法还是p[i]从1开始

如果i<=maxRight：

说明i之前被访问过了，看i关于pos对称的j,根据回文串性质p[i]至少也得是p[j]的长度，p[i]=p[j],但是不总是这样，一旦p[j]的半径右边超过了pos就不一定成立了，超过了pos那p[i]只好是自己到已经访问过的极限距离,也就是maxRight-i。

所以刚才那两个取最小的。min(p[j],maxRight-i)