**题目描述: Given a string *S*, find the longest palindromic substring in *S*. You may assume that the maximum length of *S* is 1000, and there exists one unique longest palindromic substring.**

代码:

class Solution {

public:

string longestPalindrome(string s) {

int length = s.size();

string str;

for (int i = 0, k = 0; i<length + 1; i++){ //给字符串添加#

s.insert(k, "#"); //字符串类可以直接使用下标插入(而不必使用迭代器)

k = k + 2;

}

length = length \* 2 + 1;//添加#后字符串长度

int \*rad = new int[length]();**//添加括号则rad的所有元素的初值为零**

for (int i = 1, k; i < length-1; i=i+k) {

//向两边拓展

int j=1;

while (i - j >= 0 && i + j <= length - 1 && s[i - j] == s[i + j]){

rad[i]++;

j++;

k同时也可以用来统计在半径范围内满足①②条件的i的右边元素(i+k)的个数,在下次循环时i可以从i+k开始(因为之前元素的半径根据Manacher算法已经求出)

}

//以i为对称中心的元素的半径

for (k = 1; k <= rad[i] && rad[i - k]!=rad[i] - k; k++) {

//求rad[i+k]

if (rad[i - k]<rad[i] - k)rad[i + k] = rad[i - k];

else rad[i + k] = rad[i] - k;

}

}

//寻找最大半径

int index = 0,max=0;

for (int i = 0; i < length&&s[i]; i++) {

if (rad[i]>max) {

max = rad[i];

index=i;

}

}

//处理'#'

for (int i = index - rad[index]; i < index+rad[index]+1; i++) {

if (s[i]!= '#')str.push\_back(s[i]);

}

return str;

}

};

算法主要思想:

寻找字符串中的回文，有特定的算法来解决，也是本文的主题：Manacher算法，其时间复杂度为O(n)。首先在每两个相邻字符中间插入一个分隔符，当然这个分隔符要在原串中没有出现过。一般可以用‘#’分隔。这样就非常巧妙的将奇数长度回文串与偶数长度回文串统一起来考虑了。然后，我们需要一个辅助数组rad[]，用rad[i]表示第i个字符的回文半径，rad[i]的最小值为1，即只有一个字符的情况，现在问题转变成如何求出rad数组。假设现在求出了rad[1, ..., i]，现在要求后面的rad值，再假设现在有个指针k，从1循环到rad[i]，试图通过某些手段来求出[i + 1, i + rad[i] - 1]的rad值，其分析如下：如图1所示，黑色的部分是一个回文子串，两段红色的区间对称相等。因为之前已经求出了rad[i - k]，所以可以避免一些重复的查找和判断，有3种情况：

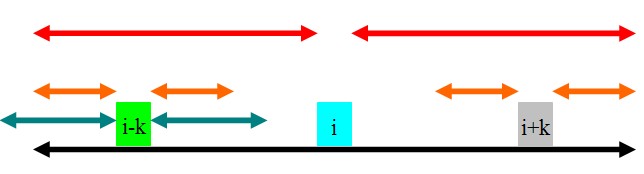


图1

**①　 rad[i] - k < rad[i - k]**

如图1，rad[i - k]的范围为青色。因为黑色的部分是回文的,且青色的部分超过了黑色的部分，所以rad[i + k]肯定至少为rad[i]-k,即橙色的部分。那橙色以外的部分就不是了吗?这是肯定的，因为如果橙色以外的部分也是回文的，那么根据青色和红色部分的关系，可以证明黑色部分再往外延伸一点也是一个回文子串,这肯定是不可能的，因此rad[i + k] = rad[i] - k。

**②　 rad[i] - k > rad[i - k]**

如图2，rad[i-k]的范围为青色，因为黑色的部分是回文的，且青色的部分在黑色的部分里面，根据定义，很容易得出:rad[i + k] = rad[i - k]。根据上面两种情况，可以得出结论:当rad[i] - k != rad[i - k]的时候,rad[i + k] = min(rad[i] - k, rad[i - k])。

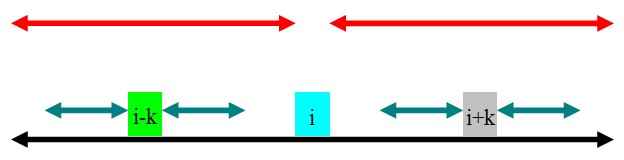


图2

**③　 rad[i] - k = rad[i - k]**

如图，通过和第一种情况对比之后会发现，因为青色的部分没有超出黑色的部分，所以即使橙色的部分全等，也无法像第一种情况一样引出矛盾，因此橙色的部分是有可能全等的。但是，根据已知的信息，我们不知道橙色的部分是多长，因此就需要再去尝试和判断了。

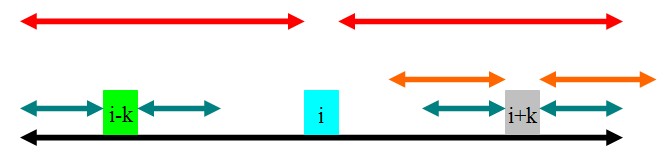


图3