4.2.3 使用Manacher算法求最长回文子串

吴永辉

Email： yhwu@fudan.edu.cn

WeChat：13817360465

ICPC Asia Programming Contest 1st Training Committee – Chair

4.2.3 使用Manacher算法求最长回文子串

- 给出一个字符串，要求计算出这一字符串的最长回文子串的长度。如果遍历每一个字符，并以该字符为中心向两边查找，则其时间复杂度为$O(n^2)$。
- Manacher算法，又被戏称为“马拉车”算法，可以在时间复杂度为$O(n)$的情况下求解一个字符串的最长回文子串的长度。

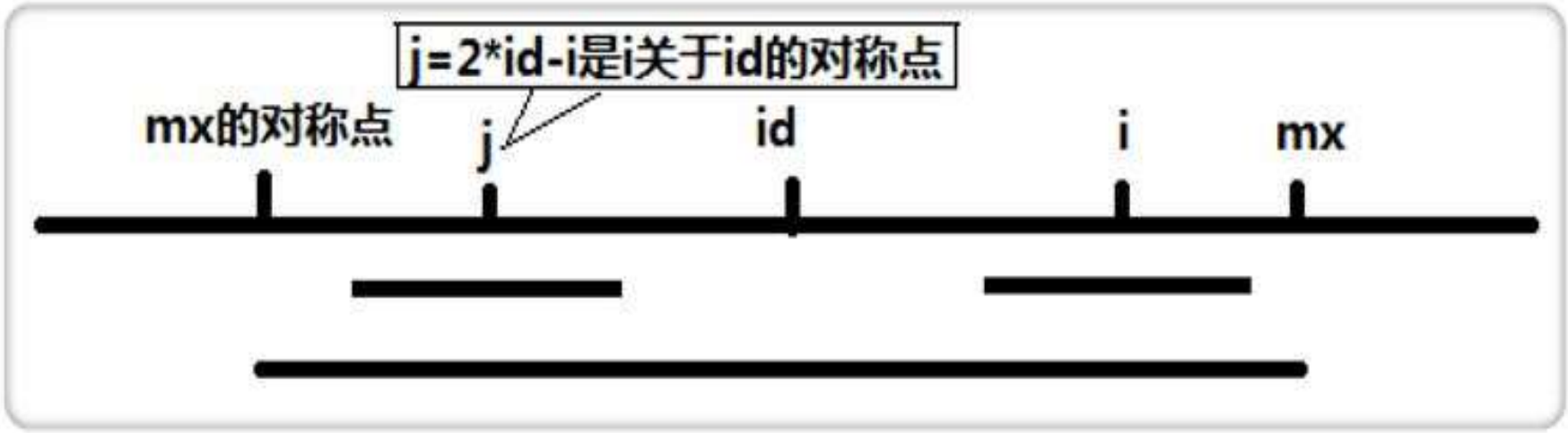
- 由于回文分为偶回文（例如"bccb"）和奇回文（例如"bcacb"），而在处理奇偶问题上会比较繁琐，例如，对于偶回文"bccb"，其对称中心是在两个‘c’字符之间；对于奇回文"bcacb"，对称中心就是‘a’字符。对此，Manacher算法在字符串首尾，及各字符间各插入一个字符，而这个字符并未出现在字符串里。例如，字符串*s*是"abbahopxpo"，用未出现在字符串里的‘#’字符插入，得新字符串*s_new*是"$#a#b#b#a#h#o#p#x#p#o#"，其中，‘$’是为了防止越界。在字符串*S*中，有一个偶回文"abba"和一个奇回文"opxpo"，分别被转换为"#a#b#b#a#"和"#o#p#x#p#o#"，回文的长度都成了奇数。

对于给出新字符串 *s_new*，定义一个辅助数组 int *p*[]，其中 *p*[*i*]表示以第 *i* 个字符为中心的最长回文的半径，如上例：

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
s_new[*i*]	$	#	a	#	b	#	b	#	a	#	h	#	o	#	p	#	x	#	p	#	o	#
p[*i*]		1	2	1	2	5	2	1	2	1	2	1	2	1	2	1	6	1	2	1	2	1

可以看出，*p*[*i*] - 1 是原字符串中以该字符所在位置为中心的回文串的长度。

对于数组 p，设置两个位置变量 mx 和 id，其中 mx 表示以 id 为中心的最长回文的右边界，即，$mx=id+p[id]$。如图所示。



- 对于$p[i]$，如果$i<mx$，设j是i关于id对称点，如图所示，则基于以下三种情况，可以求出$p[i]$的值：
- （1）以j为中心的回文串有一部分在以id为中心的回文串之外。因为mx是以id为中心的最长回文的右边界，所以以i为中心的回文串不可能会有字符在以id为中心的回文串之外；否则mx就不是以id为中心的最长回文的右边界。所以，在这种情况下，$p[i]=mx–i$。
- （2）以j为中心的回文串全部在以id为中心的回文串的内部，则$p[i]=p[j]$，而且$p[i]$不可能再增加。
- （3）以j为中心的回文串的左端正好与以id为中心的回文串的左端重合。则$p[i]=p[j]$或$p[i]=mx–i$，并且$p[i]$还有可能会继续增加，即while ($s\_new[i-p[i]]==s\_new[i+p[i]]$) $p[i]$++;
- 所以，if ($i < mx$) $p[i] = min(p[2 * id - i], mx- i)$；其中$2*id - i$为$i$关于$id$的对称点，即上面的$j$点，而$p[j]$表示以$j$为中心的最长回文半径，因此可以利用$p[j]$来加快求解$p[i]$。

4.2.3.1 Palindrome

- 试题来源：**Seventh ACM Egyptian National Programming Contest**
- 在线测试：**POJ 3974**

- Andy是一个聪明夫人的计算机专业的学生，他正在上算法课，教授问学生一个简单的问题：“你们能不能提出一个有效的算法，来找出一个字符串中最长回文的长度？”
- 如果一个字符串向前读和向后读是相同的，则称其为回文，例如"madam"是回文，而"acm"则不是回文。

- 同学们认识到这是一个经典的问题，但是他们无法找到一个比遍历所有子串并检查它们是否是回文更好的解决方案，但显然这样的算法根本没有效率。过了一会儿，Andy举手说：“OK！我有一个更好的算法。”在Andy开始解释他的想法之前，停了一会儿，然后说：“好吧，我有一个更好的算法！”
- 如果您认为您知道Andy的最终解决方案，就证明它。给定一个最多1000000个字符的字符串，请您查找并输出该字符串中最长回文的长度。

- 输入
- 您的程序将对最多达30个的测试用例进行测试，每个测试用例在一行中以最多1000000个小写字符的字符串形式给出。输入以字符串"END"开头的一行结束（为了清楚起见，用引号）。
- 输出
- 对于输入中的每个测试用例，输出测试用例编号和最长回文的长度。

试题解析

- 本题直接使用Manacher算法解题。
- 设原串*str*，长度为*L*，在原串的基础上产生辅助串*str_new*：*str_new*是在*str*的首尾及各字符间各插入一个字符'#’，*str_new*的首字符为'@'，尾字符为'$'；以第*i*个字符为中心的最长回文的半径为*p*[*i*]，初始时为0；以*id*为中心的最长回文的右边界为*mx*，即*mx*=*id*+*p*[*id*]，初始时为0；最长回文的半径为*ans*，初始时为0。

- 计算*ans*的方法如下：
- 枚举辅助串*str_new*的每一个字符位置*i*（1≤*i*≤2***L*+1），尝试该位置作为中心位置：
  - 若*mx*在*i*的右侧（*mx*>*i*），则*p*[*i*]调整为*min*(*mx*-*i*, *p*[*di**2-*i*])；否则调整为1；
  - 以*i*位置为中心计算最长回文的半径*p*[*i*]（for(; *str_new*[*i*-*p*[*i*]]==*str_new*[*i*+*p*[*i*]]; *p*[*i*]++)）；
  - 若其右边界（*p*[*i*]+*i*）大于*mx*，则中心*id*调整为*i*，重新计算右边界*mx*（*mx*=*p*[*i*]+*i*, *di*=*i*;）；
  - *ans*调整为*max*(*ans*, *p*[*i*])。
- 枚举完*str_new*的每一个字符位置后，*ans*-1即为最长回文的半径。

4.2.3.2 Best Reward

- 试题来源：**2010 ACM-ICPC Multi-University Training Contest（18）——Host by TJU**
- 在线测试：**HDOJ 3613**

- 经过一场艰苦的战斗，李将军取得了巨大的胜利。现在，国家元首决定用荣誉和财宝来奖励他所做的伟大贡献。
- 奖励李将军的一件财宝是一条由26种不同的宝石组成的项链，项链的长度为*n*，也就是*n*颗宝石串在一起构成了这条项链，而每颗宝石是属于26种宝石中的一种。
- 按照传统的观点，项链是有价值的当且仅当项链是回文——项链在任何方向上看起来都一样。然而，这个项链可能一开始还不是回文。所以国家元首决定把项链切成两半，然后把它们都交给李将军。

- 同一种类的所有宝石的价值是相同的（因为宝石的质量，宝石的价值可能是正的，也可能是负的；有些种类的宝石很漂亮，而有些宝石则看起来是普通的石头）。回文项链的价值等于宝石的价值之和，而不是回文的项链的价值为零。
- 现在的问题是，如何切割给出的项链，使两个项链的价值之和最大，并输出这个值。

- 输入
- 输入的第一行是一个整数T（$1 \le T \le 10$），表示测试用例的数量。然后给出这些测试用例的描述。
- 对于每个测试用例，第一行是26个整数：$v_1, v_2, \ldots, v_{26}$（$-100 \le v_i \le 100$，$1 \le i \le 26$），表示每种宝石的价值。
- 每个测试用例的第二行是由字符'a'到'z'组成的字符串，表示项链，不同的字符表示不同的宝石，'a'的价值是v_1，'b'的值是v_2，……，以此类推。字符串的长度不超过500000。

- 输出
- 输出一个整数，李将军可以从这串项链中获得的最大值。

试题解析

- 本题题意：对于字母表的26个字母，每个字母都有一个价值。给出一个由字母组成的字符串，将该字符串切成两份，对于每一份，如果是回文串，就计算该子串的所有字符的价值之和，否则该子串的价值为0。请您求出将字符串切成两份后能够获得的最大价值。
- 本题求解步骤如下：首先，用Manacher算法求出以每个字符为中心的回文串的长度；然后，枚举切割点，得到两个子串，由此确定每个子串的中心点；检查以该子串的中心点作为中心点的回文串的长度，如果回文串的长度等于该子串的长度，则该子串的所有字符的价值之和加入两个项链的价值之和；并对所有的两个项链的价值之和最大值。

- 计算方法如下：
- 首先，设原字符串*s*，长度为*L*；如上题，在原串的基础上产生辅助串*s_new*：然后用Manacher算法求出以每个字符为中心的回文串半径序列*p*[]，并计算原串*s*中以每个字符为尾的前缀价值和序列*sum*[]。

- 然后枚举原字符串中每个可能的切割点i（$0 \le i \le L-1$），得到两个子串$s_{0\ldots i}$和$s_{i+1\ldots L-1}$：
  - 对于左子串$s_{0\ldots i}$，在辅助串中的中心点为第$i+2$个字符，若$p[i+2]==i+2$，则左子串为回文串，其价值和为$sum[i]$；
  - 对于右子串$s_{i+1\ldots L-1}$，在辅助串中的中心点为第$i+L+2$个字符，若$p[i+L+2]==L-i$，则右子串为回文串，其价值和为$sum[L-1]-sum[i]$；

- 这样，我们就可以计算出切割点为*i*时的价值和，然后和当前最优价值*Mx*比较，调整*Mx=max*{*Mx*，切割点为*i*时的价值和}。显然，枚举完所有可能的分割点后，得出的最优价值*Mx*便是问题的解。