# 回文子串：

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

这个可以通过关于推出来；

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],但是吧不总是这样，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)

# 上升子序列：

## 方法一：

传统动态规划，复杂度n^2

## 方法二：

二分法+单调队列，例题看洛谷1020，设计算法时候，注意二分的等于号问题

新方法思路，记录一个数组tail ，这个tail实际就是单调队列

递增子序列：

tail[i],表示长度为i的最小末尾，二分这个递增数组，有=就是不减数组

递减子序列：

tail[i],表示长度为i的最大末尾，二分这个递减数组，有=就是不增数组

注意等号问题

这中有个结论，仔细想想，一个序列最长递增/递减子序列是len，那么必然非最长的递增子序列 len-1 len-2 len-3….. 1  
  
假设存在一个序列d[1..9] = 2 1 5 3 6 4 8 9 7，可以看出来它的LIS长度为5。  
下面一步一步试着找出它。  
我们定义一个序列B[1..9]，然后令 i = 1 to 9 逐个考察这个序列。  
此外，我们用一个变量Len来记录现在最长算到多少了  
  
首先，把d[1]有序地放到B里，令B[1] = 2，就是说当只有1一个数字2的时候，长度为1的LIS的最小末尾是2。这时Len=1  
  
然后，把d[2]有序地放到B里，令B[1] = 1，就是说长度为1的LIS的最小末尾是1，d[1]=2已经没用了，很容易理解吧。这时Len=1  
  
接着，d[3] = 5，d[3]>B[1]，所以令B[1+1]=B[2]=d[3]=5，就是说长度为2的LIS的最小末尾是5，很容易理解吧。这时候B[1..2] = 1, 5，Len＝2  
  
再来，d[4] = 3，它正好加在1,5之间，放在1的位置显然不合适，因为1小于3，长度为1的LIS最小末尾应该是1，这样很容易推知，长度为2的LIS最小末尾是3，于是可以把5淘汰掉，这时候B[1..2] = 1, 3，Len = 2  
  
继续，d[5] = 6，它在3后面，因为B[2] = 3, 而6在3后面，于是很容易可以推知B[3] = 6, 这时B[1..3] = 1, 3, 6，还是很容易理解吧？ Len = 3 了噢。  
  
第6个, d[6] = 4，你看它在3和6之间，于是我们就可以把6替换掉，得到B[3] = 4。B[1..3] = 1, 3, 4， Len继续等于3  
  
第7个, d[7] = 8，它很大，比4大，嗯。于是B[4] = 8。Len变成4了  
  
第8个, d[8] = 9，得到B[5] = 9，嗯。Len继续增大，到5了。  
  
最后一个, d[9] = 7，它在B[3] = 4和B[4] = 8之间，所以我们知道，最新的B[4] =7，B[1..5] = 1, 3, 4, 7, 9，Len = 5。  
  
于是我们知道了LIS的长度为5。  
  
!!!!! 注意。这个1,3,4,7,9不是LIS，它只是存储的对应长度LIS的最小末尾。有了这个末尾，我们就可以一个一个地插入数据。虽然最后一个d[9] = 7更新进去对于这组数据没有什么意义，但是如果后面再出现两个数字 8 和 9，那么就可以把8更新到d[5], 9更新到d[6]，得出LIS的长度为6。  
  
然后应该发现一件事情了：在B中插入数据是有序的，而且是进行替换而不需要挪动——也就是说，我们可以使用二分查找，将每一个数字的插入时间优化到O(logN)~~~~~于是算法的时间复杂度就降低到了O(NlogN)～！

对于递增子序列:

递增子序列：

tail[i],表示长度为i的最小末尾，如果是不减子序列，找末尾要加等号

递减子序列：

tail[i],表示长度为i的最大末尾，如果是不增子序列，找末尾要加等号

注意等号问题

2 1 5 3 6 4 8 9 7，找递增子序列·

设tail[i]是最小末尾

* 对于2:tail[1]=2，就是说当只有1一个数字2的时候，添加长度为1的LIS的最小末尾是2,tail.len=1
* 对于1：从tail第一个逐个比较,1<2,则把长度为1的最小末尾更新成了1,tail[1]=1,tail.len=1
* 对于5: 从tail第一个逐个比较,5>1,添加长度为2的最小末尾是5,

tail[1]=1,tail[2]=5;tail.len=2

* 对于3，从tail第一个逐个比较,3>1,3<5则把长度为2的最小末尾更新成3,

tail[1]=1,tail[2]=3;tail.len=2

* 对于6，从tail第一个逐个比较,6>1,6>3添加长度为3的最小末尾是6,

tail[1]=1,tail[2]=3,tail[3]=6;tail.len=3

* 对于4，从tail第一个逐个比较,4>1,4>3,4<6则把长度为3的最小末尾更新成4,

tail[1]=1,tail[2]=3,tail[3]=4;tail.len=3

* 对于8，从tail第一个逐个比较,8>1,8>3,8>6,添加长度为4的最小末尾是8,

tail[1]=1,tail[2]=3,tail[3]=4,tail[4]=8;tail.len=4

* 对于9，从tail第一个逐个比较,9>1,9>3,9>6,9>8添加长度为5的最小末尾是9,

tail[1]=1,tail[2]=3,tail[3]=4,tail[4]=8,tail[5]=9;tail.len=5

* 对于7，从tail第一个逐个比较,7>1,7>3,7>6,7<8,,则把长度为4的最小末尾更新成7,

tail[1]=1,tail[2]=3,tail[3]=4,tail[4]=7,tail[5]=9;tail.len=5

结束后，输出tail.len即是最长递增子序列的长度，如果是不减子序列则在判断大小时候，要有等于的判断；

刚才是从前往后找的，复杂度n\*n,优化到nlogn的方法是二分找tail数组，因为tail数组一定是有序的，单调队列用二分优化，复杂度就成了nlogn。

# 公共子序列问题：

## 方法一：

传统动态规划，复杂度n^2

## 方法二：

转换成最长递增子序列，自然就可以用二分+单调队列来优化时间。

转换是有条件的，要求2个序列中，某个序列是互不相同的数字组成设它是A，另一个是B，B的字符集必须被A包含，这样就能建立A的索引，ind[i]代表A数组某个等于i的数的下标(数字太大可以离散化)，把B的字符用索引去替换(B的字符集被A的包含了)，求索引的后的B序列最长不减子序列即可。正确性是由于,索引代表A每个数的位置，在B里替换后，ind[B]的递增子序列就代表了B和A按顺序重合部分。