water problem

WerKeyTom_FTD

November 25, 2017

前言

一些简单的水题。

题目来源：ceoi2016 match
给你一个长度为n的小写字母串s，请你构造一个字典序最小的括号串满足：
1,这个括号串合法（每个左括号能找到一个右括号匹配）。
2，对于两个匹配的位置i和j，$s_i = s_j$。
$n \leq 1000000$。

算法一

按字典序搜索破译串并检验即可。

算法二

用单调栈贪心做出一个合法对应括号序，可以证明任意合法解都能变改为这个括号序。
因此可以用这个方法判断可行解。

算法二

用单调栈贪心做出一个合法对应括号序，可以证明任意合法解都能变改为这个括号序。
因此可以用这个方法判断可行解。
接下来可以暴力尝试扭转右括号，并继续判断余下部分是否有解。
复杂度$O(n^2)$。

算法三

我们设$f(x,y)=1$表示区间[x,y]存在合法括号序，否则表示不存在。
这个f具备可加可减性，即
若$f(i,k)=1$，$f(k+1,j)=1$则$f(i,j)=1$，减法也是一样。
我们希望实现一个过程，每次找到一个位置在破译串中对应匹配的位置，那么剩余可以分治进行。
假设找的是i对应的位置pos，有$pos=\max\{x$满足$s[1]==s[x]$且$f(x+1,n)=1\}$
假若设$dp(i,c)=\max\{j$满足$f(j+1,i)=1$且$s[j]=c\}$，那么预先处理出这个数组，便可以快速找到pos。

算法四

我们给出一个结论：如果$f(x,y)=1$那么在位置$x-1$的单调栈和在位置y的完全一致。
于是可以对各个位置的单调栈用hash或trie上节点进行表示。

算法四

我们给出一个结论：如果$f(x, y) = 1$那么在位置$x - 1$的单调栈和在位置y的完全一致。
于是可以对各个位置的单调栈用hash或trie上节点进行表示。
然后考虑$solve(l, r)$表示处理出这个区间的破译串。
显然要找到l的pos，再递归。
对于同一哈希值同一字符，它显然是单调的，也就是这个pos在减小。
然后如果我们优先$solve(pos + 1, r)$，就可以让所有的指针都到$[l, pos]$，于是继续递归$solve(l + 1, pos - 1)$。
然后我们就线性解决了本题。

感谢wxh010910。
一个长度为n的序列，将它随机打乱，顺序放入一个队列中，然后将k个元素出队并加入小根堆中。接下来执行以下事件。
1，若堆不为空，则出堆一个元素x，如果这是第i次出堆，$ans+=b_i*x$。若堆为空，结束过程。
2，若队列不为空，则出队一个元素并将该元素加入堆中。
请求出ans的期望值。
$n \le 50$。
$n \le 300$。
$n \le 2000$。
当任意i满足$b_i=n-i+1$时$n \le 1000000$。
每个范围都存在解法。

算法一

直接模拟题意暴力。O(n!*n log n)。

算法二

我们不妨枚举一个数x考虑其的贡献。
然后有个很经典的思路是把$\leq x$的当作0，$> x$的当作1，对整个01序列进行dp。
设$f(i,j,k,0/1)$表示做到第i个位置，已经使用了j个0，其中小根堆中有k个0，第四维的含义是，如果0表示还没有遇到x，否则表示遇到了x。
这个dp很好转移，也很好统计答案，你可以在某一个合适的状态发现x出堆了，剩余部分大概是个组合数。复杂度$O(n^4)$。

算法二

我们不妨枚举一个数x考虑其的贡献。
然后有个很经典的思路是把$\leq x$的当作0，$> x$的当作1，对整个01序列进行dp。
设$f(i,j,k,0/1)$表示做到第i个位置，已经使用了j个0，其中小根堆中有k个0，第四维的含义是，如果0表示还没有遇到x，否则表示遇到了x。
这个dp很好转移，也很好统计答案，你可以在某一个合适的状态发现x出堆了，剩余部分大概是个组合数。复杂度$O(n^4)$。
接下来我们将介绍基于该算法的两个优化，每个优化都将去掉一个n。

优化一

我们给出一个结论：在第i个位置($i \geq k$)上出堆的元素是队列前i个元素中的前$i-k+1$小。

优化一

我们给出一个结论：在第i个位置($i \geq k$)上出堆的元素是队列前i个元素中的前$i-k+1$小。
容易用归纳法证明这个结论。
那么我们原先dp的第三维不需要了，第一维和第二维已经足够帮助我们判断是否已经出堆。

优化二

考虑先给第四维加上一个2，即设成$f(i,j,k,0/1/2)$。
1的含义变成遇到了x但是x仍未出堆，2的含义是x已经出堆。

优化二

考虑先给第四维加上一个2，即设成$f(i,j,k,0/1/2)$。
1的含义变成遇到了x但是x仍未出堆，2的含义是x已经出堆。
同样我们不再记录方案，而是记录带权方案，当第四维是0/1时还表示方案，当第四维是2时表示的是每种方案乘上对应系数的和。
这样做的理由是我们最后统计的就是方案乘权值的和，好处是我们可以一直dp到最末尾。

优化二

考虑先给第四维加上一个2，即设成$f(i,j,k,0/1/2)$。
1的含义变成遇到了x但是x仍未出堆，2的含义是x已经出堆。
同样我们不再记录方案，而是记录带权方案，当第四维是0/1时还表示方案，当第四维是2时表示的是每种方案乘上对应系数的和。
这样做的理由是我们最后统计的就是方案乘权值的和，好处是我们可以一直dp到最末尾。
然后我们就可以用经典套路优化，也就是可以根据0的个数对应出唯一的x，那么x不需要枚举。

关于特殊性质

接下来我们来考虑特殊性质如何利用做到$O(n)$。

一些小结论

我们来推一下小结论。

一些小结论

我们来推一下小结论。
结论一：最大的$k-1$个数是最后才出堆的。
易证。

一些小结论

我们来推一下小结论。
结论一：最大的$k-1$个数是最后才出堆的。
易证。
结论二：如果一个数不是最大的$k-1$个数，它的出堆系数为c，那么有$c-(k-1)$种方法插入一个最小数在队列中使得它系数+1。
这个结论也很容易。这是在一个长度为n的排列中插入一个比所有元素都小的元素，之后需要利用。

线性做法

设$f[i]$表示前i大的数形成的任意排列的答案。
于是每次插一个最小的数然后考虑影响。
首先前k项的f很好算。只考虑$k+1-n$。

线性做法

设$f[i]$表示前i大的数形成的任意排列的答案。
于是每次插一个最小的数然后考虑影响。
首先前k项的f很好算。只考虑$k+1-n$。
首先$f[i]+=f[i-1]*i$，因为答案实际就是带权方案，那么现在方案数要乘i（有i种插入最小数的方案）。
然后考虑最小数插在前k个数某个数的前面，那么其系数为i，$f[i]+=(i-1)!*k*a[i]*i$
然后考虑不是插在前k个数的前面，容易发现其系数是等差数列，求和一下有$f[i]+=(i-1)!*(i-1+k)*(i-k)/2*a[i]$

最后一种转移

接下来考虑对已有数的系数影响。
注意到$f[i-1]=\sum_{allpermutation}\sum_j a[j]*j*c$
插入一个最小数后，最大的$k-1$个数系数肯定不变，我们考虑记一个tmp表示这$k-1$个数的固有贡献。
然后我们写出这样的式子。
$f[i]+=f[i-1]-(tmp+(s[i-1]-s[k-1])*(k-1))*(i-1)!$
$f[i-1]-tmp*(i-1)!$后就不含最大的$k-1$个数。

接下来考虑对已有数的系数影响。
注意到$f[i-1]=\sum_{allpermutation}\sum_j a[j]*j*c$
插入一个最小数后，最大的$k-1$个数系数肯定不变，我们考虑记一个tmp表示这$k-1$个数的固有贡献。
然后我们写出这样的式子。
$f[i]+=f[i-1]-(tmp+(s[i-1]-s[k-1])*(k-1))*(i-1)!$
$f[i-1]-tmp*(i-1)!$后就不含最大的$k-1$个数。
其余的部分减出来是这样的$\sum_{allpermutation}\sum_j a[j]*(c-(k-1))$
恰好有$c-(k-1)$种插入的方法使其系数+1，那么就统计出了系数+1的贡献。