p

可持久化后缀数据结构

陈立杰

杭州外国语学校

2013 年 2 月 8 日

专注后缀数据结构30年

大家好,我们又见面了,我是clj.

专注后缀数据结构30年

一晃一年过去了,我依然专注于后缀数据结构.

字符串问题经典解法盘点

p

暴力

哈希

后缀数组

后缀树

后缀自动机

什么是可持久化数据结构?

可持久化数据结构跟传统数据结构的不同在于,传统数据结构是破坏性的,一旦进行了修改,以前的版本就失去了,可持久化数据结构则不一样就算进行了修改,一样可以在历史的版本上询问.

什么是可持久化数据结构?

我们来举一个例子,比如我们这里有一个人,我们当然只能跟现在的他对话,可是如果我们有了时光机, 我们就可以跟他的历史版本对话.

什么是可持久化数据结构?

这有什么用呢,一个最简单的例子就是我们的Editor,很多Editor都支持任意步数的undo和redo,这其实就是在访问你文本的历史的版本.打个比方说吧,如果你在处理一个1MB的大文件.

如果你进行了几千次修改操作,Editor为了支持undo,如果每次修改都把整个保存一遍,那岂不是要几GB空间了?

这当然不可能,因为他们有高效的数据结构.

可持久化X后缀数据结构

后缀数据结构,诸如后缀树,后缀自动机,都支持一个在最后加一个字符的操作.

那么我们有没有办法让他们变得可持久化呢?

可持久化X后缀数据结构

这两者的模型,一个是树,一个是自动机,都太过于复杂,一个结构越复杂当然可持久化的难度就越大.

可持久化X后缀数据结构

但是我们注意到,后缀数组本质上只是一个后缀排序之后形成的数组,数组是一个很简单的可以可持久化的数据结构.

小普及,使用平衡树实现可持久化数组

相信大家都知道如何使用平衡树实现在某个位置插入一个数之类的操作.

我们可以使用函数式平衡树来实现可持久化数组.

由于跟主题无关不了解的人可以自主学习.

可持久化后缀数组

那么有了可持久化数组,我们来想办法实现可持久化后缀数组.

注意到如果在后面添加字符,多个后缀的大小关系可能发生变化,从而影响复杂度.

可持久化后缀数组

我们不妨变更思路,我们将整个字符串倒过来,那么就变成了在前面添加字符.

不妨设整个字符串为S,添加了一个字符c,那么实际上,我们的修改就只有添加了一个后缀cS而已.

那么说白了,我们只需要找出cS在后缀数组$SA(S)$中的插入位置,在可持久化数组中插入它既可.

可持久化后缀数组

如何寻找cS的插入位置?最直接的做法就是套用平衡树的查找操作,在平衡树上查找一遍.

这样我们就需要实现两个后缀大小的比较操作了,最直接的方式是使用哈希.

我们可以用哈希先二分求出两个后缀的LCP,然后比较,那么一次比较的复杂度是$O(\log n)$.

总共的操作复杂度就是$O(\log^2 n)$.

哈希方法的缺点

随着科学的进步,OI中传统的直接运算式哈希已经被hack了.

直接运算式哈希相当于对2^{32}或者2^{64}取模.

如果我们要稳妥的使用哈希,必须要使用一个素数为模.

因为模运算很慢,这样就会大大降低程序的速度.

可持久化后缀数组

我们换一种思路考虑问题,毕竟我们只需要比较cS和以前的一个后缀t.

考虑两种情况.

$t[0] \neq c$.那么我们直接比较$t[0], c$就能知道大小关系了.

$t[0] = c$.那么我们现在需要比较S和$t[1:], t[1:]$表示t去掉第一个字符后的串.

注意到,S和$t[1:]$都是现在就已经存在了的后缀,所以他们的大小关系是已知的.

我们只需要看一下它们的大小关系就行了,可以通过求出这两个后缀各自的位置在$O(\log n)$的时间内实现.

那么总体复杂度还是$O(\log^2 n)$

可持久化后缀数组

那么我们有了这个可持久化后缀数组,怎么来回答询问呢.

字符串数据结构回答的经典询问就是询问一个字符串在该字符串中出现了几次.

不妨设询问串是s.

我们只需要在平衡树上二分,找出第一个比串s大的后缀L.

再找出第一个比串$s\#$大的后缀R,$\#$是一个比所有字符都要大的虚字符.

那么$[L, R)$这个区间的后缀就都以s开头,那么就出现了$R - L$次.

一次比较的复杂度是$O(|s|)$.那么总复杂度就是$O(|s| \log n)$了.

可持久化后缀数组:height数组

我们知道height数组是$\mathrm{sa}[i], \mathrm{sa}[i+1]$之间的LCP.他在解决字符串问题中有一定作用.

那么我们不妨通过直接在后缀i上记录height_i来实现.

在插入一个新后缀cS的时候,我们怎么更新height呢?

可以发现有两个后缀的的height需要更新.

在哈希实现中,我们直接二分得出height既可.

在另一种实现中,我们发现我们需要计算S和$t[1:]$的LCP,这就是他们之间所有后缀的height的最小值.

后缀数组在Trie上的扩展

Trie是对线性字符串的直接拓展,相应的,我们也可以定义在Trie上的后缀数组.

假设我们有一个有N个节点的Trie,不妨令每个点的"后缀"定义为从该节点往上到根的路径经过的边上的字符依次组成的字符串.

注意到是从底下到根不是根到底下.

在这里字符串的后缀数组算法仍然适用,我们可以通过倍增算法或者DC3求出Trie的后缀数组.

后缀数组在Trie上的扩展

我们注意到,如果我们只在字符串的末尾操作,实际上我们的字符串就形成了一个Trie.

那么我们如果顺便维护这个Trie的后缀数组,实际上我们就等于维护了这个串的所有历史版本.

一个简单的例子就可以是如果我们要询问,有多少串曾经在这个串或者其历史版本中出现过,就可以使用一个平衡树来维护这个历史版本Trie的后缀数组解决问题.

后缀自动机在Trie上的扩展

我们考虑一个Trie的后缀自动机,它的目标是识别这个Trie的所有"后缀".

简单的回顾后缀自动机

一个识别所有后缀的自动机.

节点u,表示结束位置$r \in \mathrm{Right}_u$, $len \in [\mathrm{L}_u, \mathrm{R}_u]$的所有子串.

所有节点的Right集合呈树状关系,所以节点数量只有$O(n)$个.

根据以上的性质一个足够NB的人就能推理出后缀自动机的一些细节(当然不包括我).

后缀自动机在Trie上的扩展

我们来考虑在线构造后缀自动机的算法,容易发现该算法对Trie也是适用的,我们原本是调用extend($last$, w)函数在最后一个后缀对应的状态上操作,现在我们可以采用dfs或者bfs,通过对父节点调用extend来得出子节点的状态.

时间复杂度大家可以自己去思考.

Codechef April2012 TSUBSTR

我们给你一个Trie,定义它的子串为从一个点走到它的一个子孙经过的边形成的串,它的子串集合就是所有这样可能的子串.

输入中的字母是随机生成的.

我们每次给你一个26个英文字母的排序,询问根据这个排序,他的子串集合中第k大的是多少.

建出后缀自动机,剩下的就是标准的工作了.

练习题

光说不练可不行.

简单的小题目

题目简述:

给你一个字符串集合,要你支持一些操作:

在字符串集合中的一个字符串后面加一个字符放入字符串集合.

询问一个字符串在该集合中的一个字符串中出现了几次.

解法:

直接套用可持久化后缀数组轻松解决.

不简单的小题目

题目简述:

给你一个字符串,你要支持一些操作:

在字符串的开头,结尾,正中间($|s|/2$处)插入或删除一个字符.

询问一个字符串出现了几次.

不简单的小题目

慢慢分解问题,首先我们现在有一个数据结构T1,它可以支持在最后插入删除一个字符.

我们不妨构造两个T1:L, R.L是倒过来的,然后用$\mathrm{rev}(L) + R$表示整个字符串.

我们称其为T2,那么T2就可以在开头和结尾插入或删除字符了.

注意到有时候L或者R,被删完了,我们把整个字符串重新均匀的分成两个字符串.在平摊的意义上不影响复杂度.

那么询问我们就可以分别在L, R中询问,同时对于询问串s,跨越L, R的情况中只需要考虑中间的长度为$O(|s|)$的部分,做一遍KMP既可.

不简单的小题目

那么我们现在有T2了.

我们再构造两个T2:L, R.并且令L, R中间的那个位置就是要插入的正中间,每次在前后插入的时候,我们在L, R间转移字符保持这个性质.

在中间插入也是同理.

我们令这个结构为T3,那么T3就能支持我们想要的操作了.

好复杂

练习题

多做多练.

练习题

COT4

题目概述:我们有两个字符串集合P, Q.

有三个操作:

从P中复制出一个字符串,在末尾添加一个字符后加入P.

从Q中复制出2个字符串u, v,将uv加入Q.

询问一个$q \in Q$在一个$p \in P$中出现了几次.

Remember History

题目概述:我们对一个串在最末尾进行了一些操作.

操作完了之后,不妨令$\mathrm{Com}(u,v)$表示历史版本u,v的公共子串数量.相同内容但不同位置的子串算多次.

对每个历史版本u,求$\sum_v \mathrm{Com}(u,v)$.

Remember History II

题目概述:我们对一个串在最末尾进行了一些操作.

操作完了之后,不妨令$\mathrm{Com}(u,v)$表示历史版本u,v的公共子串数量.相同内容但不同位置的子串算一次.

每次询问两个历史版本u,v,计算它们的$\mathrm{Com}(u,v)$.

Remember History III

题目概述:我们对一个串在最末尾进行了一些操作.

必须在线,不妨令$\mathrm{Com}(u,v)$表示历史版本u,v的公共子串数量.相同内容但不同位置的子串算一次.

每次询问两个历史版本u,v,计算它们的$\mathrm{Com}(u,v)$.

END

那么我的讨论就结束了.大家再见.

感谢

练习题COT4来自黄嘉泰(fotile96).

其它都是原创题.

如果你有什么问题,可以访问ask.fm/WJMZBMR