字符串

skip2004

字符串的定义

字符串是由若干种字符连接而成的串。

由一些字符组成的集合我们称为字符集。常见的字符有大小写拉丁字母、数字等等。

常见的字符集有小写拉丁字母、数字、‘01‘、‘ATGC’ 等等。

字符集的种类往往与我们解题无关，我们只需要在乎字符集内的字典序，字符集的大小。

字符串的表示

全课件使用 Python 表示法，具体的，会用到以下几种表示：

$len(s)$，为 s 的长度，即字符个数（也会用 $|s|, n$ 表示），$s[l]$ 为 s 的第 $l+1$ 个字符。

$s[l:r]$ 为 s 的子串，即第 $l+1$ 到第 r 个字符组成的字符串。

$s[l:r:d]$ 为 $s[l:r]$ 每 d 个字符取一个字符（第一个字符取），事实上我们允许 $d<0$，但这里只需记得 $s[::-1]$ 为 s 的翻转。

特别的，上述所有内容中，如果 $l<0$，则将其加上 $len(s)$，r 同。

l 和 r 可以省略不写，但是 : 必须保留，l 默认为 0，r 默认为 $len(s)$（好像 $d<0$ 不是，但不用管）。

```
>>> "abcdefg"[3]
'd'
>>> "abcdefg"[2:5]
'cde'
>>> "abcdefg"[2:-1:2]
'ce'
```

字符串的表示

我们称呼 $s[:k]$ 为 s 的一个前缀，$s[k:]$ 为 s 的一个后缀。

KMP

首先我们先定义 'border'。

如果一个 $0 \leq k < n$ 满足 $s[:k] = s[n-k:]$，则其是一个 'border'。

性质一：border 的 border 是 border。

性质二：两个不同的 border 直接也有 border 关系。

性质三：border 事实上呈现一个树状的关系。

利用性质一二，我们可以轻松设计出一个求出一个串所有前缀的最长 border。

Z-algo

对所有后缀 $s[k:]$ 求 $lcp(s[k:], s)$，即最长公共前缀。

manacher

求一个串每个位置为中心的最长回文串。

Trie

Trie 是一种非常重要的字符串数据结构。

他可以存好多字符串以及它们的前缀。

Trie 每个节点上都是一个字符，一个节点代表的字符串是根到它路径上的字符拼接所得到的字符串。

Trie

如何建立一棵 trie。

考虑插入一个字符串 s，先插入这个字符串一个前缀 $s[:k]$，这时候我们考虑插入 $s[:k+1]$。

首先我们看 $s[:k]$ 所在节点有没有 $s[k]$ 这个儿子，若没有，则创建。然后我们把当前节点改成 $s[:k]$ 所在节点的 $s[k]$ 这个儿子就好了。

AC 自动机

考虑把 Trie 与 KMP 结合，搞一个可以做多串匹配的玩意儿。

回文自动机

我们不加证明的给出一个结论，一个字符串 s 本质不同的回文子串最多只有 n 个。

由此，我们可以想到对于所有回文子串，我们可以建立一个 trie。来表示所有的回文子串。

回文自动机

对于每一个节点 x，它都代表了一个回文串 s，$c[x][i]$ 代表的回文串是 $i+s+i$。

特别的，我们定义一个 xpp 串，xpp 串的长度是 -1，且 xpp 加上一个字符是空串。在一开始，回文自动机里有一个 xpp 串和空串。

一个奇长回文串必然是由一个 xpp 串在两侧插入若干个字符构成的，一个偶长回文串必然是由空串在两侧插入若干字符构成的。

我们用 $len[x]$ 表示串 x 的长度。$fail[x]$ 表示 x 节点所代表串最长回文 border 所代表的串。

回文自动机

我们思考一下，假设我们建立了 $s[:i]$ 的回文自动机，那么我们可以用一条 fail 链上的点表示所有回文后缀。我们加入字符 $s[i]$，并且从链上往上跳。

可以发现的是第一次存在两边都是 $s[i]$ 的串的时候，就是唯一可能出现新回文串的时候，也只会出现一个（也证明了最多只有 n 个本质不同回文串）。

后缀排序

我们首先可以使用倍增法求所有后缀的排名。

首先我们假设我们求出了所有 $s[i:i+2^j]$ 这些字符串的排名。我们要考虑求 $s[i:i+2^{j+1}]$ 这些字符串。

最简单的方法当然是排序，但是我们应该如何比较呢？

比较两个字符串 $s[i_0:i_0+2^{j+1}]$ 和 $s[i_1:i_1+2^{j+1}]$，我们可以先比较他们的前半部分，即 $s[i_0:i_0+2^j]$ 和 $s[i_1:i_1+2^j]$，如果他们不相同，那么可以直接调用之前求出的排名来比较，不然就可以通过 $s[i_0+2^j:i_0+2^{j+1}]$ 和 $s[i_1+2^j:i_1+2^{j+1}]$ 来比较。

后缀树

可以认为是所有前缀在 Trie 上的虚树。
可以用后缀排序建立，但事实上我们可以用后缀自动机建立。

后缀自动机

常用的后缀自动机算法可以同时建立自动机与反串的后缀树。

考虑建立后缀树。

首先，我们要有每个节点的父亲 $fail$，以及它所代表字符串的长度 mx。

为了辅助建立这个结构，我们还要有一个转移数组 c。

$c[x][i]$ 表示节点 x 所代表的的字符串在前面加入字符 i 后对应的字符串节点。

后缀自动机

我们考虑先插入短后缀，再插入长的。

如果后缀 $s[k:]$ 所代表的节点是 p，那么我们现在要插入后缀 $s[k-1:]$。

首先，我们创立一个节点 np，作为新的后缀树节点，显然的是，$mx[np] = mx[p] + 1$。

可以发现，节点 p 以及它的祖先们可能在之前时候，往前插入字符 $s[k-1]$ 并没有对应节点，现在有了，所以我们给他们加上。

当第一个有 $s[k-1]$ 这条转移边的节点 p2 出现时，可以发现 $c[p2][S[k-1]]$ 的父亲就是 np 的父亲。

后缀自动机

然而事情没有这么简单，因为插入这个节点的时候，第一次连接到后缀树上是 $q = c[p2][S[k-1]]$ 到它的父亲这条边上，所以说我们可能还要在边中间创建一个新的节点 nq。

由于创建了新的节点 nq，新节点的父亲是 $fa[q], mx[q] = mx[p2] + 1$，我们应该再修正一下 c 的转移指针。

显然的是，$c[p2][S[k-1]]$ 要改成 nq，而且在 $p2$ 到祖先这条路上，把 $S[k-1]$ 这条转移边是 q 的都改成 nq。

这样就做完了所有工作。

后缀平衡树

一种使用平衡树维护后缀字典序的数据结构。
可以在字符串前面加入字符。

卷积是一种特殊的匹配方法，我们可以使用 bitset，FFT 等方式优化。