字符串选讲

AceSrc

AceSrc@outlook.com

June 4, 2019

Overview

字符串相关定义

- 串 (string): $S = S_1 S_2 \dots S_n$, $S_i \in \Sigma$.
- 子串 (substring): $S[i, j] = S_i S_{i+1} \dots S_j$, 若不满足 $1 \le i \le j \le n$, 那么 $S[i, j]$ 表示空串 ϵ.
- 前缀 (prefix): $Pre_S(i) = S[1, i]$,
- 后缀 (suffix): $Suf_S(i) = S[i, n]$.
- border 集合: $\mathbb{B}(S) = \{Pre_S(j) | Pre_S(j) = Suf_S(n - j + 1) \wedge j \neq n\}$.
- 最长 Border: $fail(S) = \arg\max_{s \in \mathbb{B}(S)} |s|$.

因为" 我的 border 的 border 还是我的 border", 我们可以通过迭代 $fail$ 来得到 $\mathbb{B}$, 即

$$\mathbb{B}(S) = \{fail^{(i)}(S) | i \in \mathbb{N}\}$$

字符串相关定义 2

- 若 S 是 T 的子串, 那么 T 是 S 的 superstring.
- 一个串 w 是 S 的 cover 当且仅当对于 S 的每个位置 i, 总存在 S 的一个子串 $S[x, y]$ 使得 $x \leq i \leq y$ 且 $S[x, y] = w$.
- 一个串 w 是 S 的 seed 当且仅当存在一个 S 的 superstring T, 使得 w 是 T 的 cover.
- 一个整数 p 被称为 S 的周期当且仅当 $\forall 1 \leq i \leq n - p : S_i = S_{i+p}$.

Some facts

周期与 border

若 $s \in \mathbb{B}(S)$, 则 $|S| - |s|$ 是 S 的周期.
同样的, 若 $|S| - x$ 是 S 的周期, 则 $Pre_S(x)$ 是 S 的 border.

Weak Periodicity Lemma

若 p, q 是 S 的周期, 且 $p + q \leq |S|$, 则 $gcd(p, q)$ 也是 S 的周期.

Strong Periodicity Lemma

若 p, q 是 S 的周期, 且 $p + q - gcd(p, q) \leq |S|$, 则 $gcd(p, q)$ 也是 S 的周期.

Some facts 2

cover 与 border

如果 w 是 S 的 cover, 那么 w 也是 S 的 border.

cover 与 cover

如果 v 是 S 的 cover, 串 u ($|u| < |v|$) 是 S 的 cover 当且仅当 u 是 v 的 cover.

KMP

我们可以利用 KMP 算法在 $O(n)$ 时间内求解
$fail(Pre_S(1)), fail(Pre_S(2)), \ldots, fail(S)$.
因为 $fail(Pre_S(i+1)) = \arg\max_{t \in \mathbb{B}(Pre_S(i)) \wedge S_{|t|+1}=S_{i+1}} |t|$.

匹配问题

匹配问题是这样一类问题:
给定文本串 S, 模式串 T, 输出所有的 i, 使得 $S[i-|T|+1, i] = T$.
利用 fail 函数可以在 $O(|S| + |T|)$ 时间内解决普通字符串的匹配问题.
每次失配之后根据 fail 函数进行后移.

热身题

http://codeforces.com/contest/631/problem/D
解决一个匹配问题，这里的字符串是由压缩形式给出的.
比如 *3-a 2-b 4-c 3-a 2-c* 表示串 *aaabbccccaaacc*.
文本串和模式串都最多分成 20000 段.
比如对于以下输入

```
6 1
3-a 6-b 7-a 4-c 8-e 2-a
3-a
```

应输出 6.

热身题题解

首先文本串和模式串相邻字符相同的段合并，使得相邻段字符不相同.
这时候如果模式串只有 1 段，那么可以直接暴力.
否则，设模式串一共有 N 段，那么我们可以发现如果要进行匹配，也得在文本串中找出 N 段，模式串和文本串对应的段的字母要相同，对于第 1 段和第 N 段，文本串的数字部分要不小于模式串的数字部分. 中间 $N-2$ 段数字要相等.
这样我们只需要将模式串的首尾去掉，做一个简单的匹配. 找到一个匹配位置就判断一下首尾是否合法.

热身题 2

http://codeforces.com/contest/432/problem/D
求 S 的每个 border 在 S 中的出现次数.
对于

```
ABACABA
```

要输出

```
3
1 4
3 2
7 1
```

根据 fail 函数可以建出一棵树, 其中 $fail(u)$ 是 u 的父亲.
那么 S 的某个前缀在 S 中出现次数就是对应子树的大小.

每个前缀的最小 cover

http://codeforces.com/gym/100431 E

给定一个字符串 S, **求** $Pre_S(1), Pre_S(2), \ldots, Pre_S(n)$ **的最小** cover.

```
abaabaababa

1 2 3 4 5 3 4 5 3 10 3
```

每个前缀的最小 cover - 题解

- 如果 w 是 S 的 cover, 且 $w \neq S$, 那么 w 也是 $fail(S)$ 的 cover.
- 如果 w 是 $fail(S)$ 的 cover, 且 w 也是 S 的 cover, 那么所有比 w 短的 $fail(S)$ 的 cover, 也是 S 的 cover.

于是乎, 我们只需要判断 $fail(S)$ 的最小 cover w 是不是 S 的 cover, 如果是, 那么 S 的最小 cover 也是 w. 否则 S 的最小 cover 是 S.
对于每个前缀 $Pre_S(i)$, 实时维护 $rlimit[i]$ 表示 $Pre_S(i)$ 能 cover 到多远.

每个前缀的最小 cover - 代码

```
1     for (int i = 1; i <= n; i++) {
2       if (fail[i] == 0) {
3           R[i] = i;
4           rlimit[i] = i;
5       }
6       else {
7           if (rlimit[ R[ fail[i] ] ] + R[ fail[i] ]
8               >= i) {
9               R[i] = R[ fail[i] ];
10              rlimit[ R[i] ] = i;
11          }
12          else {
13              R[i] = i;
14              rlimit[i] = i;
15          }
16      }
17    }
```

all-covers-problem

All-Covers-Problem

求一个串 S 的所有 cover.

因为 cover 必定是 border, 所以 covers 数不会超过 n.

ACP - fail 树

先使用 KMP 计算出 fail 函数，然后建立 Fail 树.
那么 S 的 border 对应树上的一条链. 我们逐一判断每个 border 是否能成为 cover.
一个 border 能成为 cover，当且仅当，考虑它在 S 中所有出现位置，任意两个相邻的出现位置的差不能超过其长度.
而正如之前所说，某个前缀在 S 中的所有出现位置对应了一棵子树.
从根沿着 S border 那条链往下走，出现位置会越来越少，相邻位置的差会越来越大. 所以用一个带删除的双向链表处理一下就可以了.

Trie 树上的 Fail 函数

假设现在我们有若干单词 (words) 组成字典 (dictionary).
我们把这些单词插入到一棵 Trie 树里面, 设这棵树是 $G = (V, E)$.
这棵树的任意一个节点 u 都对应了一个字符串 $\mathbb{W}(u)$, 对应了某个单词的前缀.
记点 u 的深度是 $D(u)$.
对这棵树上的节点定义 fail 函数:

$$fail(u) = \arg\max_{v \in V \wedge \mathbb{W}(v) = Suf_{\mathbb{W}(u)}(D(u) - D(v) + 1)} |v|$$

相信大家都很熟悉 Aho-Corasick automation, 以及 fail 函数的计算.

插入

```
1  map<int, int> son;
2  void insert(const vector<int> &d) {
3    int u = root;
4    for (auto v : d) {
5      if (son[u].count(v) == 0)
6        son[u][v] = newnode();
7      u = son[u][v];
8    }
9  }
```

沿着 fail 往上跳

```
1 int go_by_fail(int u, int key) {
2   while (u != 0 && son[u].count(key) == 0)
3     u = fail[u];
4   if (u == 0) return root;
5   return son[u][key];
6 }
```

计算 fail 函数

```
1  void build() {
2    fail[root] = 0;
3    queue<int> Q;
4    Q.push(root);
5
6    while (!Q.empty()) {
7      int u = Q.top();
8      Q.pop();
9
10     for (auto [key, v] : son[u]) {
11       fail[v] = go_by_fail(fail[u], key);
12       Q.push(v);
13     }
14   }
15 }
```

在自动机上查询

```
1 void search(const vector<int> &d) {
2   int u = root;
3   for (auto &v : d) {
4     u = go_by_fail(u, v);
5     /* do something */
6   }
7 }
```

势能分析

```
1 int go_by_fail(int u, int key) {
2   while (u != 0 && son[u].count(key) == 0)
3     u = fail[u];
4   if (u == 0) return root;
5   return son[u][key];
6 }
```

对于每个点 u, 定义势能函数 $\Phi(u) = d(u) - d(fail[u])$, 其中 $d(u)$ 指 u 在 Trie 树上的深度.

对于每个点 u, 在计算其儿子节点 v 的 $fail$ 时, **while** 的执行次数小于等于 $\Phi(v) - \Phi(u)$.

于是 **while** 的执行次数不会超过叶子的深度和, 小于等于字典内所有单词的总长.

暖胃题

`http://codeforces.com/contest/710/problem/F`

强制在线维护一个字典, 支持以下操作:

- 插入一个单词
- 删除一个单词
- 给定一个串 S, 询问这个串有多少个子串 $S[i,j], 1 \leq i \leq j \leq |S|$ 属于该字典.

输入的字符串的长度和不超过 3×10^5.

暖胃题题解

假设没有删除.
维护一个 Trie 树集合 ST. 每插入一个单词, 就往 ST 中放进一棵只有这个单词的, 计算过 fail 函数的 Trie 树,
如果 St 中最小的两棵 Trie 树大小相等, 从 ST 中删除这两棵树, 同时将这两棵树内所有单词插入到同一棵树里面, 计算 fail, 再把新树放进 ST 中.
这样可以保证 ST 中始终只有 $O(\log n)$ 棵树.
回答询问只要在所有 ST 树上进行询问.
同时每个单词都只会被合并 $O(\log n)$ 次.
所以总复杂度就只有一个 $\log$.
插入单词和删除单词分开做, 出现过的单词出现次数减去删除过的单词的出现次数即为答案.

后缀树

假设我们拥有一个串 S, 将 S 的所有后缀插入到 Trie 树里, 然后缩掉那些度为 2 的非根且不代表后缀的节点.
缩完点之后树的节点数不会超过 $2n$.
得到的就是一棵可以得到每个子串在 S 中出现位置的树.
后缀自动机可以在线性时间内计算出 $S^R = S_n S_{n-1} \dots S_1$ 的后缀树.
他还有别的一些作为自动机的性质.
以下提到的后缀自动机对 S 建的树均指 S^R 的后缀树.
注意后缀树按做一个遍历就可以得到后缀数组 (先走字典序小的边.)

CF 编号是 100 倍数题.

http://codeforces.com/problemset/problem/700/E

给定一个串 S, 求一个最长的序列, $A_1, A_2, \ldots, A_m$, 使得 A_i 都是 S 的子串, 且 A_{i-1} 在 A_i 的出现次数至少为两次.

题解

容易发现, A_{i-1} 一定是 A_i 的 border.
那么, A 一定分布在 SAM 的 parent 树上一条链上的.

SAM 上任意一个节点 u, 将 u 这个点代表的最长的串记为 $M(u)$. 同时 u 的任意一个祖先 v, 一定有 v 代表的所有串在 $M(u)$ 中出现次数相同.

所以我们只需要考虑每个节点代表的最长的串 $M(u)$.

Joker

带通配符 * 的字符串叫做 string with jokers.
带通配符的字符串匹配问题叫做 matching with jokers.

怎么做?

matching with jokers - 传说中的 FFT 裸题.
记文本串 S, 模式串 T.
枚举字符 c, 将 S 转成 S^c, 定义为:

$$S_i^c = \begin{cases} 1, & S_i = c \\ 0, & S_i \neq c \end{cases}$$

将 T 转成 T^c, 定义为:

$$T_i^c = \begin{cases} 1, & S_i \neq c \wedge S_i \neq * \\ 0, & ow \end{cases}$$

求 $R_k^c = \sum_{i-j=k} S_i^c T_j^c$, 若 $R_k^c \neq 0$, 那么意味着发生失配, $i - j = k$ 的位置不能合法匹配.

更优美一点的做法

记文本串 S, 模式串 T.
将字符转化成正整数, 比如 $\underline{a} \to 1$, $\underline{b} \to 2$ ....
将 Joker 转成 0.
求 $R_k = \sum_{i-j=k} S_i(S_i - T_j)^2 T_j$.
同样, 判断 R_k 是否为 0.

单位复根

如果 jokers 只存在于模式串中.
记文本串 S, 模式串 T.
将字符转化成正整数, 比如 $\underline{a}$ - $\underline{1}$, $\underline{b}$ - $\underline{2}$ . . . .
将 Joker 转成 0.
将 S 转为 S', 定义为:

$$S'_i = e^{j\frac{s_i}{|\Sigma|}}$$

将 T 转为 T', 定义为:

$$T'_i = e^{j(-\frac{s_i}{|\Sigma|})}$$

求 $R_k = \sum_{i-j} S'_i T'_j$.
那么只有 R_k 的虚部为 0, 实部为 T 中非 joker 字符个数时, 位置 $i-j$ 能产生匹配.

没啥出的意义的题

给定一个文本串 S 和一个模式串 T, 对于 $1 \leq i \leq |S| - |T| + 1$, 求 $S[i, i + |T| - 1]$ 和 T 的 Hamming Distance.

没啥出的意义的题题解

如果字符集比较小，那么可以用之前的方法，枚举字符，然后 fft.
如果字符集比较大，那么根据每个字符在 T 中出现次数分类. 小于 S 的直接暴力更新. 大于 S 的做 fft.
时间复杂度 $O(n\sqrt{n\log n})$

中场题

https://nanti.jisuanke.com/t/A1533

对于一个模式 M 以及一个字符串 S, 如果存在一个定义域和值域都是字符集的双射函数 f, 满足以下条件, 则称 S 符合模式 M.

- $|M| = |S|$
- $\forall i \in \{0, 1, \cdots, |S| - 1\}, f(S_i) = M_i$

比如串 BCC, DEE, 都满足模式 ABB, 但 ABC, AAA 不满足.

给定若干模式, 每次给出一个询问串 S, 问是否有 S 的某个子串满足某一个模式.

只有一个模式的时候

考虑现在有模式串 M, 和文本串 S, 我们想知道是否有 S 的某个子串符合 M.
先对模式串和文本串进行转化, 变成一个整数序列. 对于某个字母, 如果是在串中第一次出现, 那么对应整数 0, 如果不是, 则对应到其上一次出现的距离.
比如串 $ABBACAB$ 对应整数序列 $0, 0, 1, 3, 0, 2, 4$.
将 0 看成有一定限制的通配符.
比如在 $ABBACAB(0, 0, 1, 3, 0, 2, 4)$ 中查找模式 $ABC(0, 0, 0)$.

只有一个模式的时候

现在我们可以定义两个整数序列同构的条件
我们称 $\bar{M} \equiv \bar{S}$ 当且仅当对于所有的 i, 以下两条至少有一条满足.

- $\bar{M}_i = \bar{S}_i$
- $(\bar{S}_i > i \vee \bar{S}_i = 0) \wedge (\bar{M}_i > i \vee \bar{M}_i = 0)$

这个 $\equiv$ 具有很重要的传递性.

模仿 KMP

类似 KMP 算法, 我们可以对整数序列 S 计算 fail 函数.
多个模式的时候利用基于 fail 函数的 AC 自动机即可.

k-LCF

k-LCF 是这样一类问题:

k-LCF

给定字符串 S, T, 求 S 的一个子串 S', T 的一个子串 T', 使得 S' 与 T' 的 Hamming 距离不超过 k, 且 $|S'| = |T'|$ 最大.

$O(nmk\log n)$

令 $|S| = n$, $|T| = m$.
显然, 我们可以通过二分 Hash 求 LCP, 以 $O(nmk\log n)$ 的复杂度来解决这个问题.
如果用复杂的后缀数组求 LCP, 就可以以 $O(nmk + (n + m))$ 的复杂度来解决这个问题.

$O(nm)$

令 $\Phi(i,j)$ 为 $S[1,i]$ 和 $T[1,j]$ 最长的, mismatches 不超过 k 的后缀长度.
枚举 $s=i-j$, 不失一般性, 令 $s \geq 0$.
想办法以此求解 $\Phi(s+1,1), \Phi(s+2,2), \ldots$
注意到, $i-\Phi(i,i-s)$ 是有单调性的, 用个队列维护一下就好了.

$k = 1$

对于 $k = 1$ 的特殊情况有更好的复杂度.
题目: [BZOJ 3145][Feyat cup 1.5]Str.
解 **1-LCF** 问题

假设失配点是 S_i, T_j, 那么答案是
$LCP(Suf_S(i+1), Suf_T(j+1)) + LCS(Pre_S(i-1), Pre_T(j-1)) - 1$.
将两个串用特殊字符拼接, 做正向的后缀数组 F 和反向的后缀数组 B.
那么上面的式子对应了 F 某个区间的最小值加上 B 某个区间的最小值再加一.
从大到小枚举 F 对应的最小值, 每次会将若干小区间合并成大区间. 对于 F 中已经合并起来的若干后缀, 只有在 B 中相邻的才有用.
那么对每个合并块维护一个平衡树, 支持查找前驱和后继即可. 可以利用启发式合并.
时间复杂度 $O(n \log^2 n)$.
该问题学术界的最优复杂度是 $O(n \log n)$. (但我还没看懂.)

A Text Problem

给定一个串 T, 有若干询问, 每次询问给出一个串 S, 问 T 有多少个子串和 S 的 Hamming 距离不超过 1.
$|T|, \sum |S| \leq 10^5$
来源: ICPCCamp 2017 Day4 K.

做法 1

对 T 建后缀自动机 $A(T)$, 以及 T^R 的自动机 $A(T^R)$.
令 S 中 mismatch 的位置是 i. 将 $S[1, i-1]$ 在 $A(T)$ 上跑, $(S[i+1, |S|])^R$ 在 $A(T^R)$ 上跑.
可以得到 $S[1, i-1]$, 和 $(S[i+1, |S|])$ 在 T 中分别的出现位置, 分别对应了两个自动机 Parent 树一棵子树内的节点, 需要求交.
求出欧拉序, 变成平面二维数点.
离线处理二维数点的话复杂度是 $O(|T||\Sigma| + \log|T|(\sum|S| + |T|))$.

做法 2

对 T 建后缀数组.
令 S 中 mismatch 的位置是 i, 使用二分维护出以 $S[i+1,|S|]$ 为前缀的后缀在后缀数组内的位置, 这一定是个区间 $[L,R]$.
枚举 mismatch 的字符 c, 用二分维护出以 $S[1,i-1]+c$ 为前缀的后缀在后缀数组内的位置, 这也是个区间 $[L',R']$.
利用这两个区间, 利用二分可以得到 $S[1,i-1]+c+S[i+1,|S|]$ 在 T 中出现位置.
复杂度 $O(|T|+|\Sigma|(\sum|S|)\log|T|)$.

Indeterminate Matching

http://www.51nod.com/Challenge/Problem.html#!#problemId=1532

给定一个文本串 S, 和一个模式 T.

T_i 是 Σ 的一个子集, 表示这一位可接受的字符.

比如 $T_i = \{a, b, c\}$, 表示第 i 位可以接受的字符是 $\underline{a}$, $\underline{b}$, $\underline{c}$.

输出所有匹配位置.

输入:

```
aaaabacabcabd
3
3 abc
2 bc
3 abc
```

输出:

```
4
6
8
```

直接使用 Shift-And.
先算出对于每个字符 x, 维护一个 bitset, 记录他在模式串中的出现位置.
然后枚举文本串位置 i, 维护一个 bitset 记录 $S[i-j..i]$ 是否能匹配 $T[1,1+j]$.
每次转移只需要移位和与操作.

求循环串的所有 cover

给定一个循环串 S, 求其所有的 cover.

求循环串的所有 cover

给定一个循环串 S, 求其所有的 cover.
首先将 S 复制一遍, 对 SS 建后缀自动机.
利用线段树合并我们可以在 $O(n \log n)$ 知道 SS 每个子串出现位置相邻差的最大值.
再根据第一次出现位置和最后一次出现位置判断一下即可.

求串的所有 seed

给定一个串 S, seed 的个数, 这里的 seed 需要是 S 的子串.

求串的所有 seed

给定一个串 S, seed 的个数, 这里的 seed 需要是 S 的子串.
对 S 建后缀树.
利用线段树合并我们可以在 $O(n \log n)$ 知道 S 每个子串出现位置相邻差的最大值.
可以直接通过逆串的 fail 函数, 知道能不能覆盖尾部.
由于不能枚举子串, 但我们发现, 如果一个子串是 $S[i, j]$, 如果有 $|fail(S[1, j])| \geq i$, 那么就可以覆盖头部.
相当于想知道, 给定一个数组, 对于某个区间里面求不小于某个数的元素有多少个.
这个可以离线之后直接树状数组做.