KUPC2020 spring J: 接頭辞分解

原案, 作成: yamunaku

定義など

文字列 y をいくつかの部分文字列に分割して、その部分文字列すべてが文字列 x の接頭辞となるようにできるとき、「x は y を分解する」と言うことにする。

整数 $l, r (0 \leq l \leq r \leq |S| - 1)$ について、S の l 文字目から r 文字目までの部分文字列を $S[l : r]$ と表すことにする。

アルゴリズム

次のようなアルゴリズムを考える。

1. 文字列 $m_0 = S[0 : 0]$ とする。
2. k を 1 から $|S| - 1$ まで動かしながら、以下を行う。
   - m_{k-1} が $S[0 : k]$ を分解するとき、文字列 m_k を m_{k-1} とする。
   - m_{k-1} が $S[0 : k]$ を分解しないとき、$S[x : x + |m_{k-1}| - 1] = m_{k-1}$ となるような最大の x $(0 \leq x \leq k - |m_{k-1}|)$ を求め、文字列 m_k を $S[x : k]$ とする。

$m_{|S|-1}$ が求める答えであることが示せる。

証明

$V_0 = \{S[0 : 0]\}$ とする。また、 V_{k-1} の中で長さが最小となるある要素 m_{k-1} について、

- m_{k-1} が $S[0 : k]$ を分解するとき、$V_k = V_{k-1}$
- そうでないとき、$V_k = \{S[l : k] \mid \exists r \, (l \leq r < k), \ S[l : r] \in V_{k-1}\}$

とする。すべての $k (0 \leq k \leq N - 1)$ について、以下の (1) から (6) が成立することが示せる。

(1) V_k の長さが最小となる要素はただ 1 つ存在する。

(2) $k > 0$ ならば、m_{k-1} は m_k の接頭辞である。

(3) m_k は $S[0 : k]$ を分解する。

(4) 任意の $p_k \in V_k$ について、 m_k は p_k に含まれ、 m_k は p_k を分解する。

(5) $S[0 : k]$ を分解する任意の文字列 T について、$p_k \in V_k$ が存在して、 p_k は T の接頭辞となる。

(6) m_k は $S[0 : k]$ を分解する文字列の中で長さが最小となるものである。

次のような流れで証明する。

1. (1) を帰納的に示す。
2. (2),(3),(4) を同時に帰納的に示す。
3. (4) を用いて、(5) を帰納的に示す。
4. (3),(5) を用いて、(6) を示す。

1. (1) の証明

V_k の作り方から、V_k は 1 つ以上の文字列を含み、V_k が含む文字列の長さはすべて異なる。したがって、V_k の長さが最小となる要素はただ 1 つである。

2. (2),(3),(4) の証明

$k = 0$ のとき、(2),(3),(4) は成立する。

(2)　$k = 0$ だから。
(3)　$m_0 = S[0:0]$ だから。
(4)　V_0 の要素は $m_0 = S[0:0]$ だけだから。

$k = k' - 1$ のときの (2),(3),(4) を仮定し、$k = k'$ のときの (2),(3),(4) を証明する。以下では、視認性のために k' を k に置き換えている。

(2) の証明

m_{k-1} が $S[0:k]$ を分解するときは $m_k = m_{k-1}$ なので成立。

m_{k-1} が $S[0:k]$ を分解しないときは、m_k のある接頭辞 pm が存在して、pm は V_{k-1} の要素となる。$k-1$ における (4) の仮定から、pm は m_{k-1} を含む。つまり、m_k は m_{k-1} を含む。したがって、m_{k-1} は m_k の接頭辞でなければならない。なぜなら、もしそうでないとすると、 m_k の m_{k-1} が現れる部分以降を抜き出して作った文字列が V_k に含まれることになり、m_k の長さが V_k 内の文字列の中で最小であることに矛盾する。

(3) の証明

$|m_k| = l_k$ とおく。m_{k-1} は $S[0:k-1]$ を分解するから、$S[0:k-l_k]$ も分解する。また、$S[k-l_k+1:k] = m_k$ である。上で証明した k における (2) より、m_{k-1} は m_k の接頭辞なのだから、m_k は $S[0:k]$ を分解する。

(4) の証明

$|m_k| = l_k$ とおく。m_{k-1} が $S[0:k]$ を分解するときは $V_k = V_{k-1}$ なので成立。

m_{k-1} が $S[0:k]$ を分解しないとき、$p_k = S[x:k] \in V_k$ は次の 3 つの場合に分けられる。

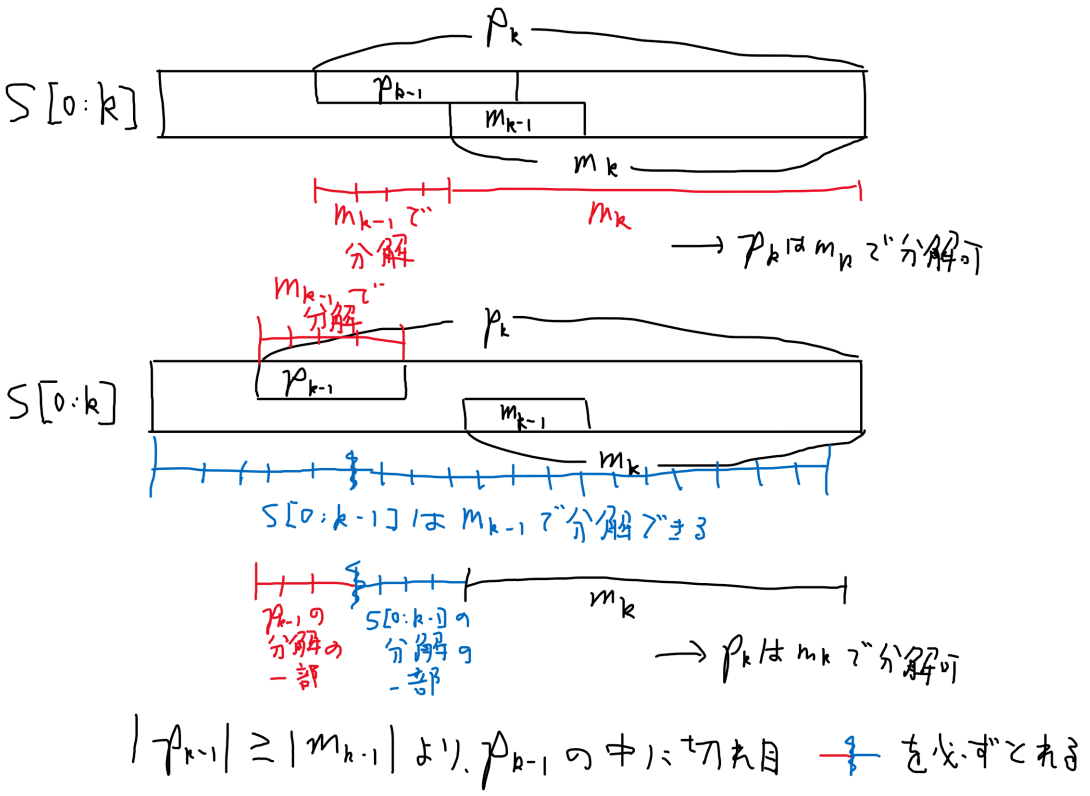
(a)　$l_{k-1} < x$
(b)　$k - l_k < x \leq l_{k-1}$

(c)　$x \leq k - l_k$

(a) のとき、m_{k-1} は $S[0:k-1]$ を分解するから、$S[0:x-1]$ も分解する。また、$S[x:k]$ は m_{k-1} の接頭辞である。したがって、m_{k-1} が $S[0:k]$ を分解できてしまうことになり、仮定に反する。

(b) のとき、$k-1$ における (4) の仮定から p_k は m_{k-1} を含む。この p_k から m_{k-1} が現れる部分以降を抜き出して作った文字列は、V_k に含まれる。これは m_k の長さが V_k の中で最小であることに矛盾する。

したがって、(c) の場合しかありえない。$k-1$ における (3) の仮定より、m_{k-1} は $S[0:k-1]$ を分解する。上で証明した k における (2) より、m_{k-1} は m_k の接頭辞である。また V_k の作り方から、ある $p_{k-1} \in V_{k-1}$ が存在して、p_{k-1} は p_k の接頭辞である。$k-1$ における (4) の仮定より、p_{k-1} は m_{k-1} を含み、m_{k-1} は p_{k-1} を分解する。これらから、m_k が p_k を分解できることを示せる。以下の図のようにして分解できる。



3. (5) の証明

$k = 0$ のとき、$S[0:0]$ を分解する文字列の先頭の文字は $S[0]$ でなければならないから、(5) は成立する。

$k = k' - 1$ のときの (5) を仮定し、$k = k'$ のときの (5) を証明する。以下では、視認性のために k' を k に置き換えている。

m_{k-1} が $S[0:k]$ を分解するときは $V_k = V_{k-1}$ である。$S[0:k]$ を分解する文字列は $S[0:k-1]$ も分解するので成立する。

m_{k-1} が $S[0:k]$ を分解しないとき、$S[0:k]$ を分解するような文字列 t を好きにとる。t は $S[0:k-1]$ を分解するので、(5) の仮定より、$p_{k-1} \in V_{k-1}$ が存在して、 p_{k-1} は t の接頭辞となる。$S[0:k]$ をいくつ

かの部分文字列に分解して、できた部分文字列すべてが t の接頭辞になったとする。分解してできた部分文字列のうち、最後のものに注目し、これを $S[x:k]$ とおく。定義から $S[x:k]$ は t の接頭辞である。x について、次の 2 つの場合に分けることができる。

(a) $k - |p_{k-1}| < x$
(b) $x \leq k - |p_{k-1}|$

p_{k-1} は t の接頭辞だったから、(a) のとき、$S[x:k]$ は p_{k-1} の接頭辞である。すると、(4) から、$S[x:k]$ は m_{k-1} で分解できることになる。$S[0:x-1]$ は m_{k-1} で分解できていたのだから、$S[0:k]$ も m_{k-1} で分解できてしまい、矛盾する。

したがって、(b) の場合しかありえない。このとき、p_{k-1} は $S[x:k]$ の接頭辞となるから、$S[x:k] \in V_k$ でなければならない。したがって $p_k \in V_k$ が存在して、 p_k は t の接頭辞となる。

4. (6) の証明

(3) より、m_k は $S[0:k]$ を分解する。(5) より、$S[0:k]$ を分解する任意の文字列 t について、$p_k \in V_k$ が存在して、 p_k は t の接頭辞となるから、$|t| \geq |p_k| \geq |m_k|$ が成立する。したがって、 m_k は $S[0:k]$ を分解する文字列の中で長さが最小となるものである。

実装

上記のアルゴリズムは、Suffix Array + Longest Common Prefix Array および Segment Tree を用いて高速に実装できる。計算量は、$O(\,|S|(\log|S|)^2\,)$ である。