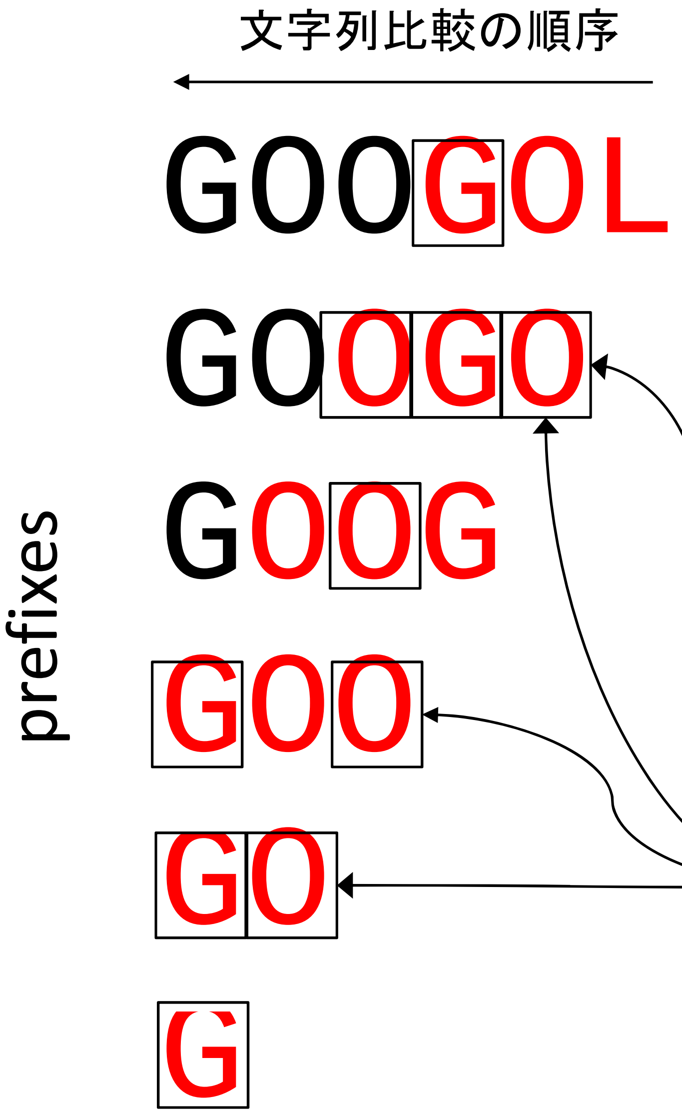
Li-Durbin Algorithm の解説

森下

2009/11/11

Searching for LOL in GOOGOL



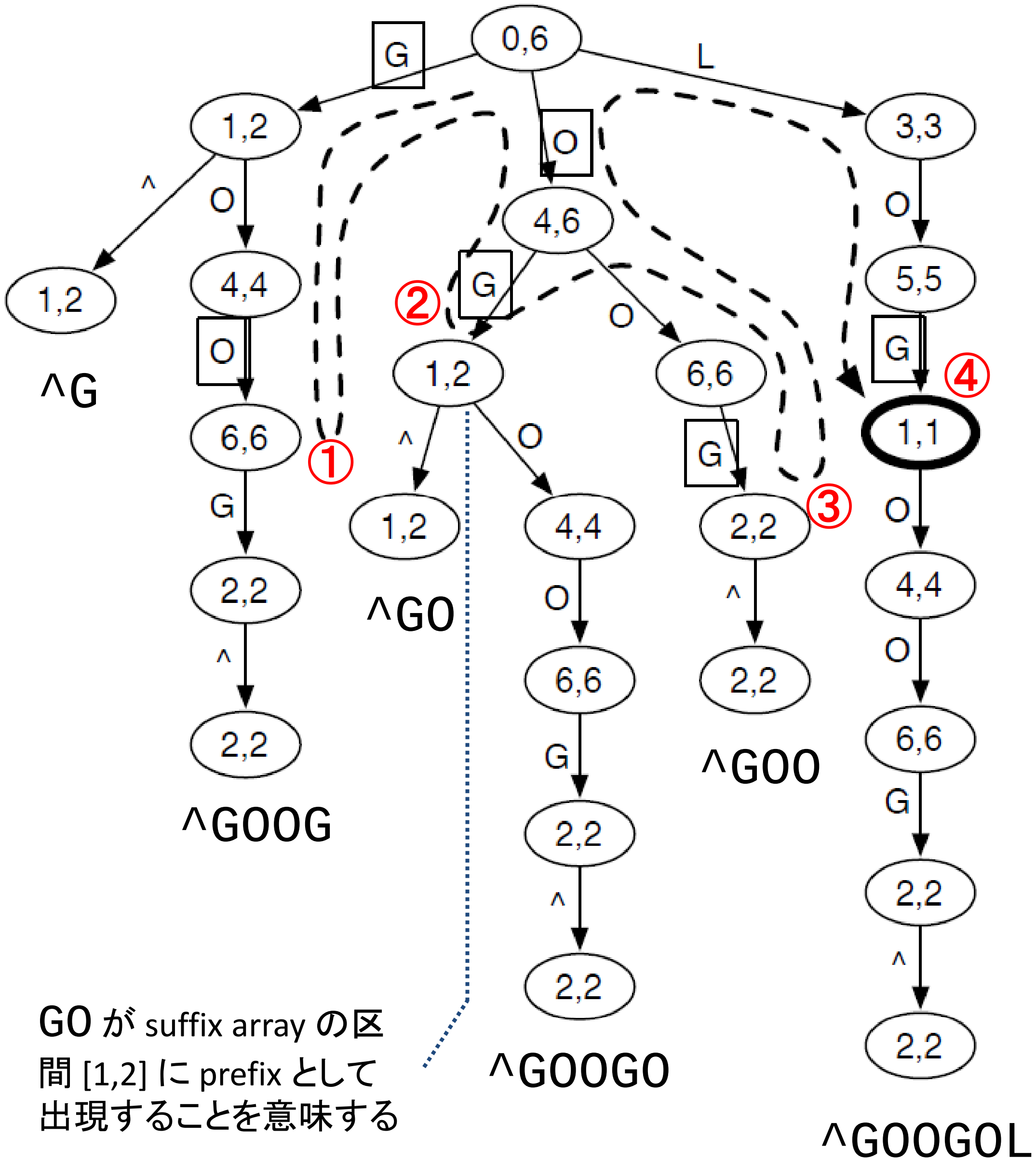
問合せの LOL が出現する場所を後から前へすべて調べ、ミスマッチ数を調べる方式 → 無駄が多い

ミスマッチ数が3。
ミスマッチ数1の3文字が他にある。
他の3文字を調べたあと、ここを調べるなら、全文字を調べなくてもよさそう。

途中の文字まで同じ。まとめられない？

□ ミスマッチ

Prefix trie: Prefix 中の文字を後から前へ順番に、
根から葉へ配置し、prefix をまとめた木構造



Suffix Array

Burrows Wheeler Transform

	Suffix Array		Burrows Wheeler Transform
0	6	$googo	l
1	3	gol$go	o
2	0	googol	$
3	5	l$goog	o
4	2	ogol$g	o
5	4	ol$goo	g
6	1	oogol$	g

Bounded Traversal and Backtracking

① LOL のOOG とのミスマッチ数は 2
② GO とのミスマッチ数は少なくとも 2 で、この下を調べても無駄（Bounded Traversal と呼ぶ）
③ GOO とミスマッチ数は 2
④ GOL とのミスマッチ数は 1

Suffix array の中では、prefix が自然に束ねられている

夏学期の試験問題の問題3(4)

S=**TATAATAATATAATA$**, W=**TAA**

	SA		BWT	A	C	G	T
0	15	$TATAATAATATAAT	A	1	0	0	0
1	14	A$TATAATAATATAA	T	1	0	0	1
2	11	AATA$TATAATAATA	T	1	0	0	2
3	3	AATAATATAATA$TA	T	1	0	0	3
4	6	AATATAATA$TATAA	T	1	0	0	4
5	12	ATA$TATAATAATAT	A	2	0	0	4
6	9	ATAATA$TATAATAA	T	2	0	0	5
7	1	ATAATAATATAATA$	T	2	0	0	6
8	4	ATAATATAATA$TAT	A	3	0	0	6
9	7	ATATAATA$TATAAT	A	4	0	0	6
10	13	TA$TATAATAATATA	A	5	0	0	6
11	10	TAATA$TATAATAAT	A	6	0	0	6
12	2	TAATAATATAATA$T	A	7	0	0	6
13	5	TAATATAATA$TATA	A	8	0	0	6
14	8	TATAATA$TATAATA	A	9	0	0	6
15	0	TATAATAATATAATA	$				

$	A	C	G	T
0	1	10	10	10

Proposition:

$lb(xW) = C(x) + Occ(x, lb(W) - 1)$

$ub(xW) = C(x) + Occ(x, ub(W)) - 1$

$lb(\{\}) = 0$

$lb(\mathbf{A}) = C(\mathbf{A}) + Occ(\mathbf{A}, lb(\{\}) - 1) = 1$

$lb(\mathbf{AA}) = C(\mathbf{A}) + Occ(\mathbf{A}, lb(\mathbf{A}) - 1) = 2$

$lb(\mathbf{TAA}) = C(\mathbf{T}) + Occ(\mathbf{A}, lb(\mathbf{AA}) - 1) = \mathbf{11}$

$ub(\{\}) = 15$

$ub(\mathbf{A}) = C(\mathbf{A}) + Occ(\mathbf{A}, ub(\{\})) - 1 = 9$

$ub(\mathbf{AA}) = C(\mathbf{A}) + Occ(\mathbf{A}, ub(\mathbf{A})) - 1 = 5$

$ub(\mathbf{TAA}) = C(\mathbf{T}) + Occ(\mathbf{T}, ub(\mathbf{AA})) - 1 = \mathbf{13}$

TAA 中の文字を後から前に処理して範囲を絞っていることに注意。

Precalculation:

Calculate BWT string B for reference string X
Calculate array $C(\cdot)$ and $O(\cdot,\cdot)$ from B
Calculate BWT string B' for the reverse reference
Calculate array $O'(\cdot,\cdot)$ from B'

- 検索対象の配列 X から BWT B と C, Occ を計算する。
- X を反転した配列から BWT B' と C', Occ' を計算する。実は C' と C は同一なのでC' は生成不要 。

Procedures:

$\textsc{InexactSearch}(W, z)$
  $\textsc{CalculateD}(W)$
  return $\textsc{InexRecur}(W, |W|-1, z, 1, |X|-1)$

- 問合せ配列 W がミスマッチ数を最大 z 個許して配列 X 中に出現する位置をすべて suffix array の区間として枚挙する。
- 問合せの部分配列 $W[0,i]$ が、X 中に最小 c 個のミスマッチで出現するとき、c の下限 $D(i)$ ($\leq$ c) を1つ計算。$D(i)$ =0 では自明なので、できるだけ大きくしたい。

- ミスマッチ許容数 z 個以下で、問合せ配列 W の0から $|W|$-1番目の文字が、X の 1から$|X|$-1の範囲に出現する位置を suffix array の区間として枚挙する。
- その際、$D(i)$ の情報をうまく利用して効率的な Bounded Traversal を実現する。

- 部分配列 $W[0,i]$ が X 中に最小 c 個のミスマッチで出現するとき、c の下限 $D(i)$ ($\leq$ c) を1つ計算する。このアルゴリズムが計算する $D(i) <$ c の例を右に示す。$D(i)$ =0 では自明なので、できるだけ大きくしたい。

例
X = AAACCCCCGGG
W = AAAGGG
X と W の最小ミスマッチ数は3だが、アルゴリズムは $D(5)$=1 <3 を計算

$$
\begin{array}{l}
\textsc{CalculateD}(W) \\
\quad k \leftarrow 0 \\
\quad l \leftarrow |X| - 1 \\
\quad z \leftarrow 0 \qquad j \leftarrow 0 \quad (j \text{ は理解のため}) \\
\quad \boxed{\begin{array}{l} \textbf{for } i = 0 \textbf{ to } |W| - 1 \textbf{ do} \\ \quad k \leftarrow C(W[i]) + O'(W[i], k-1) \cancel{+1} \\ \quad l \leftarrow C(W[i]) + O'(W[i], l) -1 \end{array}} \\
\qquad \textbf{if } k > l \textbf{ then} \\
\qquad\quad k \leftarrow 0 \\
\qquad\quad l \leftarrow |X| - 1 \\
\qquad\quad z \leftarrow z + 1 \qquad j \leftarrow i+1 \\
\qquad D(i) \leftarrow z
\end{array}
$$

- X の反転配列全域で$[k, l]$ を初期化する。
- 部分配列 $W[j, i]$ が完全マッチで出現する suffix array の区間を計算。
- 部分配列 $W[j, i]$ の i を1つ1つ増やし、$[k,l]$ の範囲を絞っている。
- $k > l$ の時はミスマッチのため$W[j, i]$ が出現しないことを意味。$j = i + 1$から始まる部分配列 $W[j, i]$ を、X 全域から検索する。
- ミスマッチ数 z を1 増やす。

- X を逆順にした配列の BWT を使っている。W の方は逆順にしなくてよいのか？
- 実は for 文の中で $W[0]$, $W[1]$, … の順番で $[k, l]$ の範囲を絞っており、W を逆順に後から前に処理していることになっている。このように X と W の文字の処理方向を合わしている。
- なぜこんな面倒なことをするのか？　部分配列 $W[0,i]$ の最小ミスマッチ数の下限を計算するのに、i = 0,1,2,3, … と単調に増やし、$O(|W|)$ で計算できる。このエ夫をしないで、X の BWT だけから $O(|W|)$ で簡潔に計算できるかどうか、考えてみてください。

$(z < 0 \,\|\, (0 <= i \;\&\&\; z < D[i]))$

$\textsc{InexRecur}(W, i, z, k, l)$
  if $\cancel{z < D(i)}$ **then**
    return $\emptyset$
  if $i < 0$ **then**
    return $\{[k, l]\}$
  $I \leftarrow \emptyset$
  $I \leftarrow I \cup \textsc{InexRecur}(W, i-1, z-1, k, l)$
  for each $b \in \{A, C, G, T\}$ **do**
  $k1 \leftarrow C(b) + O(b, k-1) \cancel{+1}$
  $l1 \leftarrow C(b) + O(b, l)$ -1
    if $k1 < l1$ **then**
      $I \leftarrow I \cup \textsc{InexRecur}(W, i, z-1, k1, l1)$
      if $b = W[i]$ **then**
        $I \leftarrow I \cup \textsc{InexRecur}(W, i-1, z, k1, l1)$
      else
        $I \leftarrow I \cup \textsc{InexRecur}(W, i-1, z-1, k1, l1)$
  return I

int $k1, l1$;　k, l だと b=C,G,T のとき引数の値が失われる

- ミスマッチ数許容数 z 個以下で $W[0, i]$ が出現する X 中の位置を、suffix array の区間 $[k, l]$ の中に探す。
- Backward search を実装するため、i の初期値を $|W|$-1 で呼び、i を減らし、0 に至るまで再帰的に呼びだす。
- ミスマッチ数の下限 $D(i)$ が許容数 z を上回った場合、条件を満たす位置は存在しない
- 探索を終了し、区間 $[k, l]$ を返す
- $W[i]$ は削除されたとみなす
- W を後から前に処理しているので、この順番でよい
- $W[i]$ は挿入されたとみなす
- マッチしている場合、許容数 z はそのまま
- ミスマッチしている場合、許容数 z は1減らす

註）　重複を除く∪の実装が難しい時は、重複を含んでいて構いません