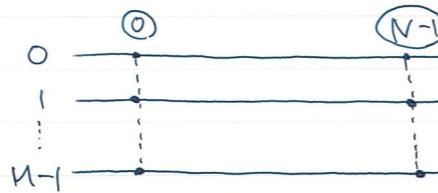


## Positional Burrows-Wheeler Transform (PBWT)

- M: haplotypeの数.  $x_i$  ( $i=0, 1, 2, \dots, M-1$ ): haplotype

N: variant sitesの数.  $k=0, 1, 2, \dots, N-1$



- 各variant siteは biallelic である.  $x_i[k] \in \{0, 1\}$

- 任意の配列  $s = s[1, 2, s[k_1, k_2]$  が  $k_1 < k_2$  のとき  $s[k_1, k_2]$  を含む配列となる。

-  $s \in t^k$  "  $k_1, k_2$  が match である"  $\Leftrightarrow s[k_1, k_2] = t[k_1, k_2]$

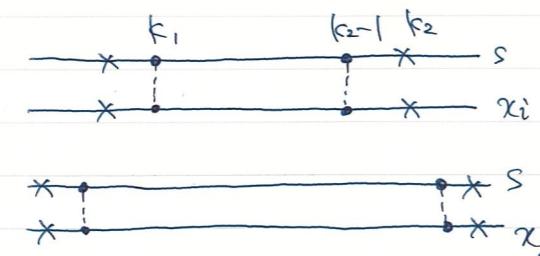


-  $s \in t^k$  "  $k_1, k_2$  が locally maximal match である"

$\Leftrightarrow [k_1 \text{ が } T \text{ の } k_1 \text{ 位置 or } s[k_1-1] \neq t[k_1-1]] \text{ and } [k_2 \text{ が } T \text{ の } k_2 \text{ 位置 or } s[k_2] \neq t[k_2]]$

-  $s \in t^k$  "  $k_1, k_2$  が set-maximal match である"

def  
 $\Leftrightarrow s \in x_i^k$  "  $k_1, k_2$  が locally maximal match である",  $x_i$  と  $x_j$  は  $s = s[k_1, k_2]$  を含む  
長い配列と match する二つある。



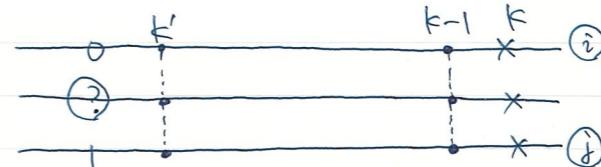
- 各variant site  $k$  ( $k=0, 1, 2, \dots, N$ ) が  $s = s[1, 2, s[k_1, k_2]$  である。  $k_1$  は  $k$  の reversed sort である。  $k_1 \rightarrow 0$  の順序は

書かれた順序 ( $k_1, k_2, \dots, k_N$ ) と一致すると言ふ。

もし  $k_1 \rightarrow 0$  が 全く同じ配列を意味する。それは  $x_i$  の  $x_i = t$  が  $t = s$  である。

①  $k=0$  の reversed sort は 何でもよい。( $= t$  が  $t = s$  である)

(Point)  $X^M$  の 2つの配列  $[k'_1, k'_2]$  が  $[k'_1, k'_2]$  set-maximal である。 $k'_1$  は  $k'_2$  reversed sort である。 $k'_1, k'_2$  は  $X^M$  の 2つの配列  $[k'_1, k'_2]$  が  $[k'_1, k'_2]$  set-maximal である。

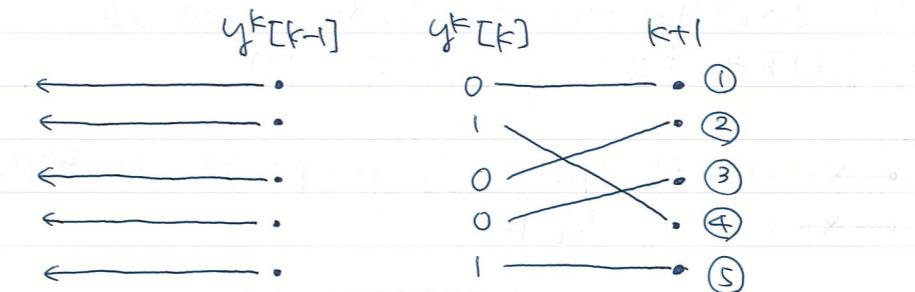


②  $k'_1$  の間に  $k'_1$  と  $k'_2$  の位置。  
③  $k'_1$  は  $k'_2$  の  $k'_1$  と  $k'_2$  が biallelic である。  
④  $k'_1$  は  $k'_2$  の  $k'_1$  と  $k'_2$  が set-maximal である。

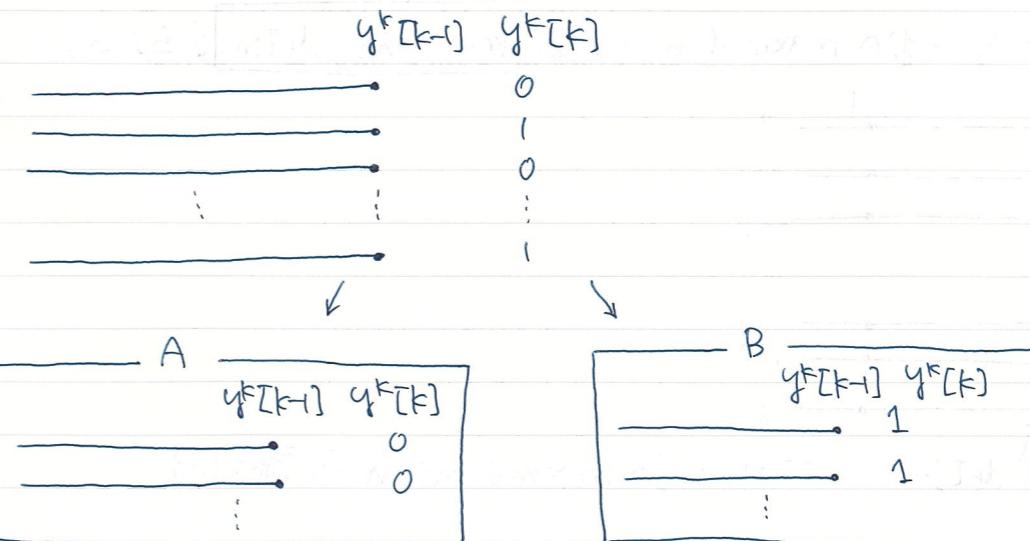
Algorithm 1  $k_1$  は  $k$  の reversed sort を求める

$a_{k[i]}$ :  $k_1$  は  $k$  の reversed sort を行う。 $t = t[i]$  は  $i$  番目に相当する  $M$  の  $X^M$  の  $i$  位置。

$y_{ik}$ :  $k_1$  は  $k$  の reversed sort を行う。 $t = t[i]$  は  $i$  番目 ( $= T$  は  $M$  の  $X^M$  の  $i$  位置)



$y^k[k]$  の値が同じである。 $a_{k+1}$  内で  $a_{k+1}$  の順序は  $a_{k+1}$  中で  $a_{k+1}$  の順序と一致する。



最後に  $A \rightarrow B$  と 変換すればいい。 $a_{k+1} = T$  は

### Algorithm 1 : Code. $a_k \rightarrow a_{k+1}$ の構成

```

 $u \leftarrow 0, v \leftarrow 0, a[], b[]$ 
for  $i=0 \rightarrow M-1$  do
    if  $y_{i,k}[k]=0$  then  $a[u] \leftarrow a_k[i], u \leftarrow u+1$ 
    else  $b[v] \leftarrow a_k[i], v \leftarrow v+1$ 

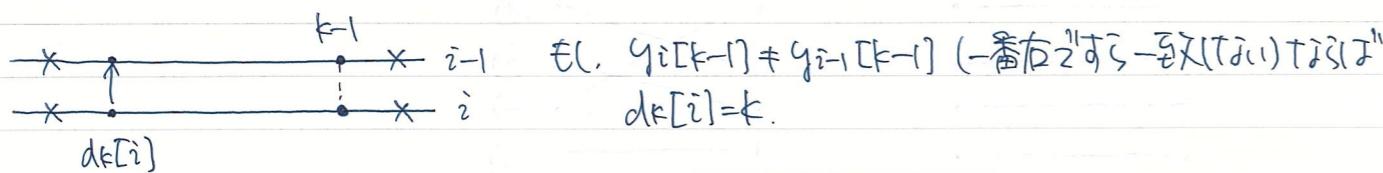
```

$a_{k+1} \leftarrow \text{Concatenate } a \rightarrow b$ .

### Algorithm 2 $a_k, d_k$ 及び $a_{k+1}, d_{k+1}$ を同時に求める

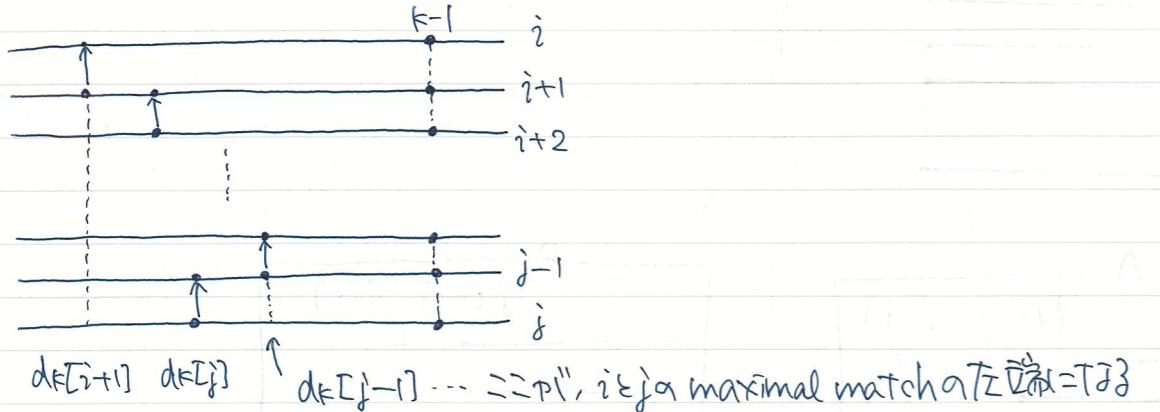
$d_k[i]$ :  $k^{\text{th}}$  reversed sort ( $T = T^k$  の  $i^{\text{th}}$  番目と、その上に "match" ある  $T^k$  の位置)

i.e.  $y_{i,j,k} = y_{i-1,j,k}$  を満たす最小の  $j$  ( $i=0, 1, 2, \dots, M$ )



同様に  $i=12$ ,  $[d_k[0] = d_k[M] = k]$  が成立する。

このとき,  $i < j \leq k$  且  $y_i \leq y_j$  が maximal match [j].  $\max_{i < m \leq j} d_k[m] \geq i$



$a_k, d_k \rightarrow a_{k+1}, d_{k+1}$  の順次求める  $T^k$  の構成

( $k=5$ )

$i \quad a_k[i]$

0	4	0, 0, 0, 0, 0, 0
1	1	1, 1, 0, 0, 0, 1
2	6	1, 1, 0, 0, 0, 1
3	0	0, 1, 0, 1, 0, 1
4	5	1, 0, 0, 0, 1, 0
5	7	0, 1, 0, 1, 1, 0
6	3	0, 1, 1, 1, 1, 0
7	2	1, 1, 1, 1, 1, 1

( $k=6$ )

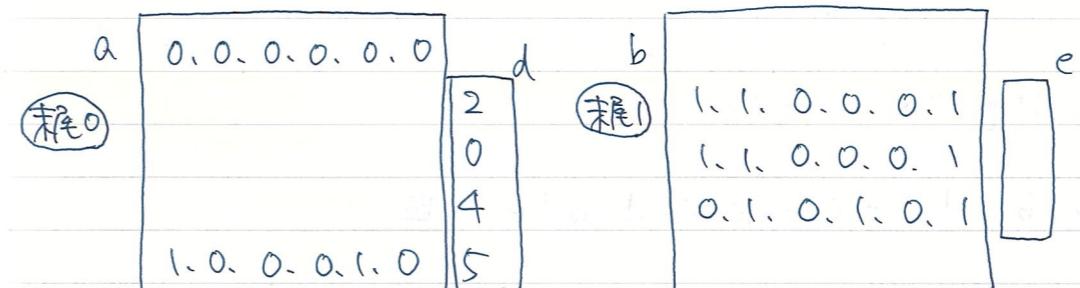
$d_k[i]$

5	→ 0, 0, 0, 0, 0, 0
2	→ 1, 0, 0, 0, 1, 0
0	→ 0, 1, 0, 1, 1, 0
4	→ 0, 1, 1, 1, 1, 0
5	→ 0, 1, 1, 1, 1, 0

$d_{k+1}[i]$

6
5
4

二の4つの中で  $d_k$  の最大のものから入る



Algorithm 1 と 同様に,  $a \oplus b = y_k^k[k]$  の直列尾 (アリタリ)  $d_k$  は  $d_{k+1}$  の直列に入れる。

$a \oplus b$  (= 配列や"入れば"そこまで add core) の中に  $\lambda, 2, 1, 3, d_k[i]$  を MaxE  $d_{k+1}$  と (採用),  $d_{\text{core}}$  のリスト (この段階で) がうまくいっている。

**Algorithm 2: Code**  $a_k, d_k \rightarrow a_{k+1}, d_{k+1}$  算法

$u \leftarrow 0, v \leftarrow 0, p \leftarrow k+1, q \leftarrow k+1, a[], b[], d[], e[]$   
 各のハコの一一番上に付ける  $d_{k+1}$  を無条件で  $d_{k+1} = k+1 (= u)$  とする

for  $i=0 \rightarrow M-1$  do:

```

if  $d_k[i] > p$  then  $p \leftarrow d_k[i]$ 
if  $d_k[i] > q$  then  $q \leftarrow d_k[i]$ 
if  $y_i[k] = 0$  then
   $a[u] \leftarrow a_k[i], d[u] \leftarrow p, u \leftarrow u+1, p \leftarrow 0$ 
```

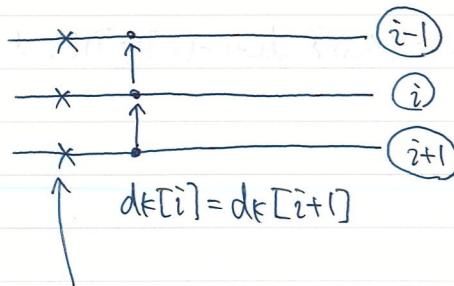
$a[]$  の  $i$  は  $a_k[i]$  を入力, 前回の  $a[]$  は入力,  $T = a_k[0]$  は付けて  $d_k[0]$  の値と LX降,  $d_k[i]$  までの最小値を  $d[u]$  に入力,  $a$  側の  $d[]$  の  $i$  は  $= i+1$  とする

else: ( $\exists j | y_j[k] = 1$ )

$b[v] \leftarrow a_k[i], e[v] \leftarrow q, v \leftarrow v+1, q \leftarrow 0$

$a_{k+1} \leftarrow \text{concatenate } a \text{ and } b, d_{k+1} \leftarrow \text{concatenate } d \text{ and } e$ .

(Point)  $d_k[i] \neq d_k[i+1]$



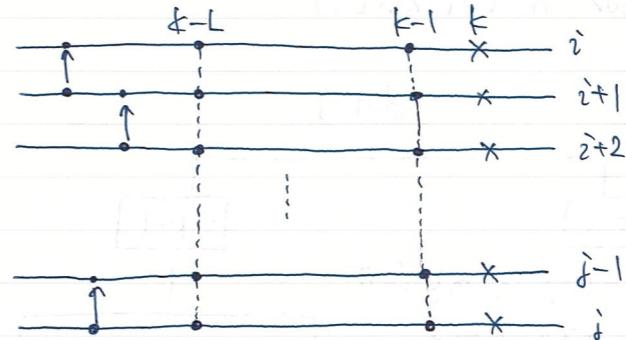
$d_k[i] - 1^2$  (2. 小2. (1負に3つの直並びがある必要がある). biallelic が2つある無理。

すべての  $k$  について  $a_k$  をまとめて positional prefix arrays とする。

**Algorithm 3**  $x_a$  中  $z^k$  が終る (=  $k-1$  まで "match" が終る)  $\sqcup$  上連続 ( $T = \text{match}$  を求める)

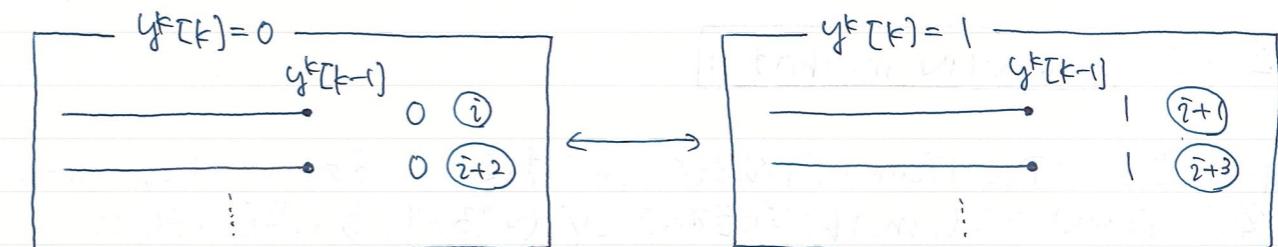
$y_i^k \in y_j^k$  は  $\sqcup$  上連続, もう  $k$  が終る  $\Rightarrow$  match をもつ  $T = k+1 = j$ ,

$y_i^k[k] \neq y_j^k[k] \Rightarrow d_k[m] \leq k-L$  ( $i < m \leq j$ )



$d_k[m] \leq k-L$  (for  $\forall i < m \leq j$ )

あとで  $a_k \rightarrow a_{k+1}$  と同じ要領で  $y_i^k[k]$  の値と共に分類を行う



両方のハコに要素が含まれるのは! 二つの組合せの一つ。  $k-1$  まで "match" が終る,  $k$  が終る,  $\sqcup$  上の長さを持つ match,  $\sqcup$  2通りある。

$\sqcup$  上で,  $a_k$  と  $d_k$  ("top, bottom") の  $\sqcup$  上の  $\sqcup$  を構成せよ;

### Algorithm3 : Code

```

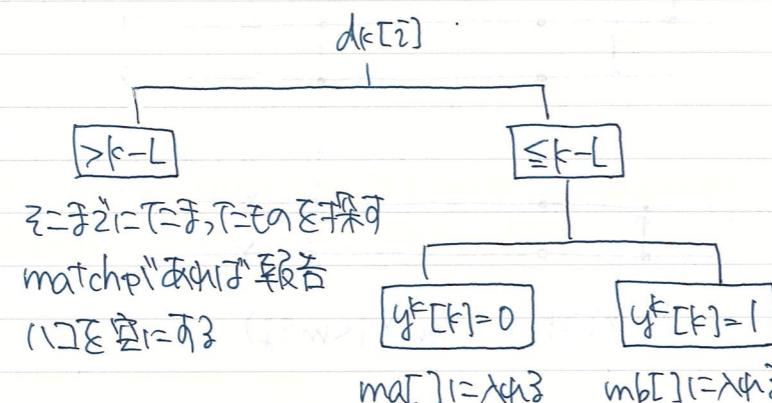
 $u \leftarrow 0, v \leftarrow 0, ma[], mb[]$ 
for  $i = 0 \rightarrow M-1$  do:
    if  $dk[i] > k-L$  then
        if  $u > 0$  and  $v > 0$  then report match for  $(ma[u], mb[v])$ 
     $u \leftarrow 0, v \leftarrow 0, ma[][], mb[]$ 

```

```

if  $y_i[k] = 0$  then
     $ma[u] \leftarrow a_k[i], u \leftarrow u+1$ 
else: ( $y_i[k] = 1$ )
     $mb[v] \leftarrow a_k[i], v \leftarrow v+1$ 

```

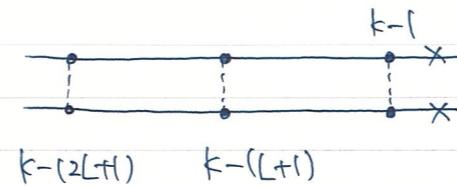


全本計算時間は  $O(\max(MN, \text{match数}))$

(外側のloopが "M個"。内側の "12回"  $k$  の範囲に限る。上記の "2" の "2"。 $k=0, 1, 2, \dots, N-12$ " 動かす。全体で  $O(MN)$ 。ただし、match 2" 報告すべき組合せが多い場合ほど多くにかかる。)

また、 $dk$  と  $y^k$  は "T 見つかる"。"k" の終点は match 列挙は可能との "2"。X の 使用量は  $O(M)$  である。

### ※ 両方向の match 検索場合



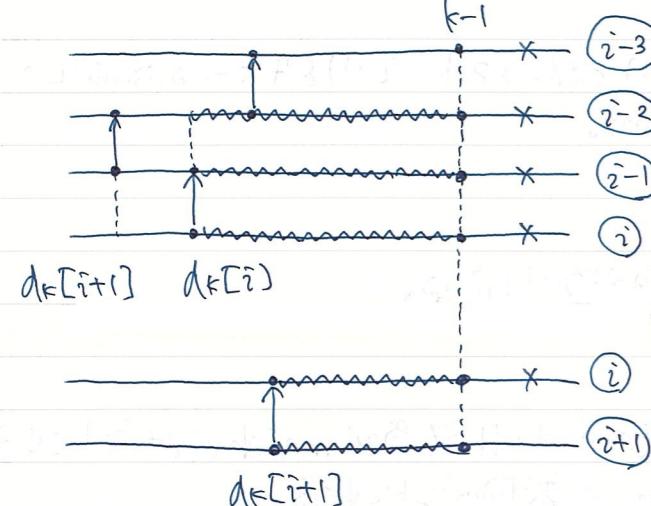
$dk[i, j] \leq k-(2L+1)$  と  $k-(L+1)$  を中央とする両方向の match を数え上げる。

$T=T+(..(本文2"2" y^k[k]の値は間かず) match 2"2" 報告する) + T, 2, 1, 3,$   
 $\Rightarrow$  すなはち、「両方向とも  $k$  以上」の match を報告する。

(例) 両方向性の match (す.例)  $k=102$  " match 12("  $T=7$  のや",  $k=11$  で再び match 2"2" と 2"2" )  
( $k=11$  で直ぐ  $y^k[k]$  を見つける)  $T=dk$ , 同 ("match 2" 複数回やうに) 2"2"

→ あくまで hit ( = match 2"2" と 2"2" ) と X の量が飛散可視化。この都度処理をいくつ (欠損 IT= 基本の imputation など?) という使い方が望ましい。

Algorithm 4.  $X_1$ において、 $k-1$ 個ある ( $k-1$  は "match" 組) set-maximal な配列を求める



この場合は、 $y_i \in F$ ,  $y_{i-1} \in F$ ,  $y_{i-2} \in F$  と最も長い match を持つ。  
 $(y_i \in F$  と  $y_{i-1} \in F$  が match を持つが、この場合は、 $y_{i-1}$  や  $y_{i-2}$  の方が "match" の長さが 2 で、 $y_i$  は set-maximal match の候補となる。)

よって、この「候補」の中から、 $y_i^k[k] = y_{\oplus}^k[k]$ となるものは①が必ずあります。matchで“ $k$ 番”も右側に近いもの、 $k$ を終わる set-maximal と配列にはない。従って、上方または下方のどちらに探索を進めるかを決定し、以下のようにします；

Algorithm 4: Code for the set-maximal match

for k=0 → N do:

$d_k[0] \leftarrow k+1, d_k[M] \leftarrow k+$

一番上と下の定義を入れる。(def[0] ← kでもいい気もするが...)

\* for  $i=0 \rightarrow M-1$  do

$$m = i - 1, n = i + 1$$

\* \* if  $d_k[i] \leq d_k[i+1]$  then

while  $dk[mt+1] \leq dk[i]$  do  $\triangleleft$   $mt+1 \leq dk[i] + 1$

if  $y_m[k] = y_i[k]$  and  $k \neq N$  next i  
 $m \leftarrow m-1$

KPL" N (last) 2"あめいげ"

$y_{m[k]} = y_i[k]$  である。つまり `match` と `def` の順序が逆である。

\* \* If  $d_k[i] \geq d_k[i+1]$  then

while  $dk[n] \leq dk[i+1]$  do マスク + (2  $dk[i+1]$ ) + 1 で。

if  $y_n[k] = y_i[k]$  and  $k \neq N$  next i  
 $n \leftarrow n + 1$

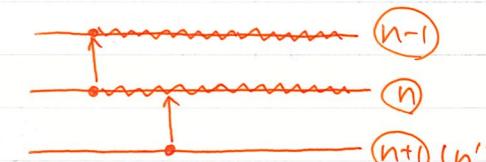
\* \* for  $j=m+1 \rightarrow i-1$  do

report match of  $\underline{A_k[i]}$  to  $A_k[j]$  from  $d_k[i]$  to  $k$ .



最後(す.  $m+1 \rightarrow m$  2"  $df[m+1] \leq df[i] + m^2$ ,  $m \neq 1$ ) 最終。そのあとに やうこトヤ" に咸り、  
 $m' (=m-1) (=t \text{ ふ } T = \text{ 次 } \text{ 要 } \text{ し } \text{ て } \text{ お } \text{ か } \text{ そ } \text{ の } 2)$  match 自体は  $m (=m'+1)$  にせぬよ。

\* \* for  $j = i+1 \rightarrow n-1$  do  
 report match of  $\underline{a_k[i]}$  to  $a_k[j]$  from  $d_k[i]$  to  $k$ .



最後は  $n \rightarrow h-1$  の  $\text{df}[n] \leq \text{df}[i+1] + j \alpha^2$ .  $n \oplus$  最終。  
 そのため  $i \oplus j \geq n' (= n+1)$  に満たす  $j \oplus k \alpha^2$   
 おける  $\text{match}$  ラストは  $n (= n'-1)$ .

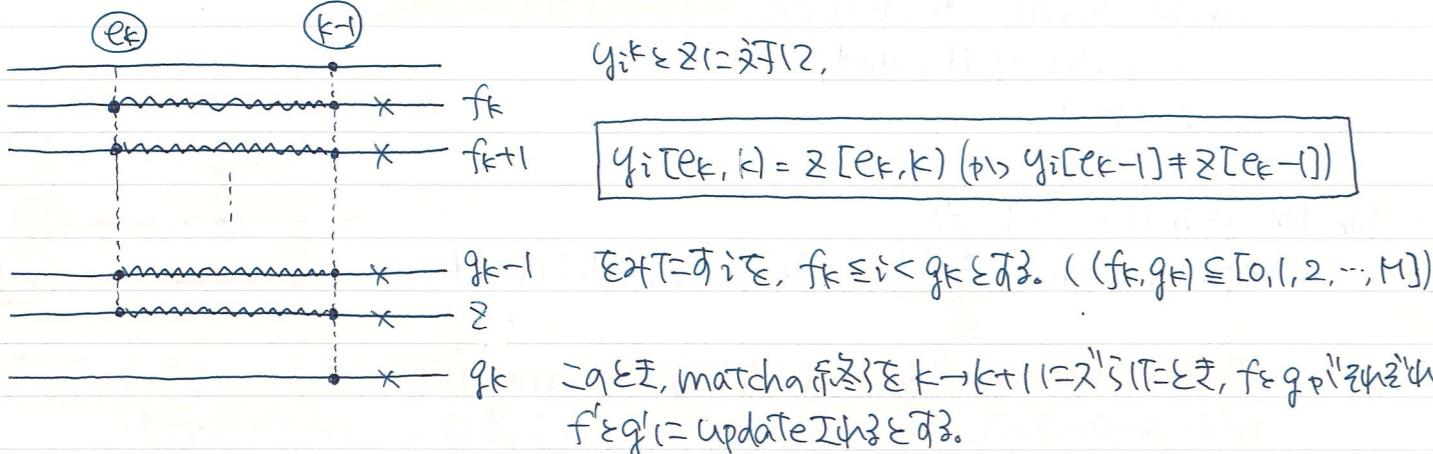
$$\begin{array}{l}
 \text{ex.} \quad 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \quad (\textcircled{s}) \\
 \begin{array}{r}
 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \quad (\textcircled{t_1}) \\
 1 \ 0 \ 1 \ 0 \ 1 \quad (\textcircled{t_2}) \\
 \hline
 \end{array} \\
 \begin{array}{c}
 \uparrow \qquad \qquad \qquad \uparrow \\
 k_1 \qquad \qquad \qquad k_2
 \end{array}
 \end{array}$$

$S \in T_2 \wedge [k_1, k_2) \subsetneq \text{dom}(T_2)$  match  $T_2 = \emptyset$ .  $S \in T_1 \wedge [k_1, k_2) \subsetneq \text{dom}(T_1)$  match  $T_1 = \emptyset$ .

辞書順に並んでるのを<sup>2</sup>。左のやうと長いmatchを示すと(マチ。よし、 $y_1 \dots y_n | y_s[k] = y_t[k]$ )  
 などやうな見出しがある。 $S$ との間に $k$ 終わるset-maximal match(マチ。次に進む $k+1$ )。

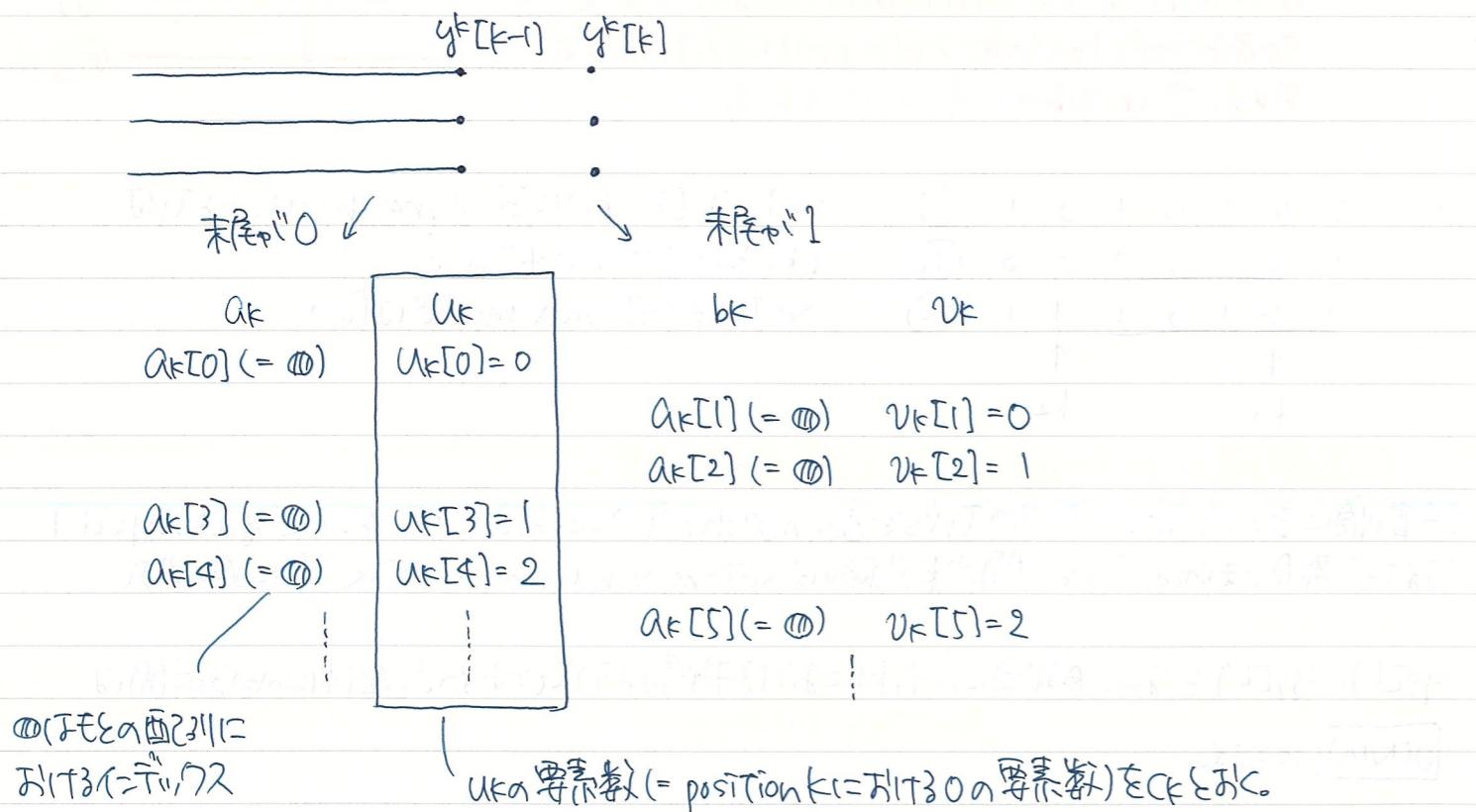
$y_i[k] = y_j[k]$  となる  $T$  の時間間に、 $(i, k)$  における探索は  $T$  で可能である。全体にかかる時間は  $O(NM)$  である。

Algorithm 5 素新(い面)の set-maximal match を求める。



$\rightarrow z[e_k, y_i[e_{k+1}, k+1]] = z[e_{k+1}, k+1]$  すなはち、 $f'_k \leq i < g'_k$  へ update する。

二つ目、Algorithm 1 の用法 (U, V を用い、 $\text{LKf}$  の値を求める)。



二つ目、 $W_k$  の値を求める。

$$\begin{aligned} W_k(i, 0) &= U_k[i] \\ W_k(i, 1) &= C_k + U_k[i] \end{aligned}$$

$W_k$  は、 $k$  における reversed sort<sup>2</sup> の番目<sup>2</sup> である<sup>2</sup>。 $k+1$  は reversed sort<sup>2</sup> の何番目に来るかを表す

$\alpha_k[i]$

$$\alpha_k[i] = \alpha_{k+1}[W_k(i, y_i[k])] \quad \cdots (*)$$

$i \leq j$  は  $\alpha_k$  の逆変換である。

つまり  $\alpha_k[i]$  は、 $k$  の reversed sort における<sup>2</sup>  $i$  番目。そして  $U_k$  の要素数<sup>2</sup> における<sup>2</sup>  $i$  番目<sup>2</sup> である<sup>2</sup>。 $k+1$  の reversed sort における<sup>2</sup>  $j$  番目<sup>2</sup> である<sup>2</sup>。つまり  $\alpha_k[i]$  は、 $U_k$  の要素数<sup>2</sup> における<sup>2</sup>  $i$  番目<sup>2</sup> である<sup>2</sup>。つまり  $\alpha_k[i]$  は、 $U_k$  の要素数<sup>2</sup> における<sup>2</sup>  $i$  番目<sup>2</sup> である<sup>2</sup>。

$$j = \alpha_k[i] \Leftrightarrow i = \alpha_k[j] \text{ である} \Rightarrow$$

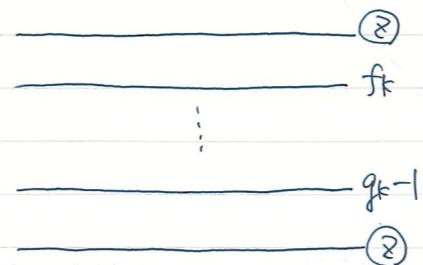
$$(*) \Leftrightarrow j = \alpha_{k+1}[W_k(\alpha_k[j]), y_j[k]] \Leftrightarrow \alpha_{k+1}[j] = W_k(\alpha_k[j], y_j[k])$$

$y_j[k+1]$  は reversed sort<sup>2</sup> の何番目?

$k$  における reversed sort<sup>2</sup> (=  $U_k$  の要素数) の番目? すなはち、 $k+1$  の reversed sort<sup>2</sup> (=  $U_{k+1}$  の要素数) の番目? すなはち、 $k+1$  の reversed sort<sup>2</sup> (=  $U_{k+1}$  の要素数) の何番目? 行うべき操作?

二つ目、 $k$  における reversed sort<sup>2</sup> の番目<sup>2</sup> である<sup>2</sup>。 $k+1$  における reversed sort<sup>2</sup> の番目<sup>2</sup> である<sup>2</sup>。 $k+1$  の reversed sort<sup>2</sup> の番目<sup>2</sup> である<sup>2</sup>。 $k+1$  の reversed sort<sup>2</sup> の番目<sup>2</sup> である<sup>2</sup>。 $f_k, g_k$  を用い、 $z[e_k, g_k]$  の途中に入ること<sup>2</sup> である<sup>2</sup>。

(Point)  $d_k[i] \neq d_k[i+1]$  のとき、 $z[e_k, g_k]$  の途中に入ること<sup>2</sup> である<sup>2</sup>。

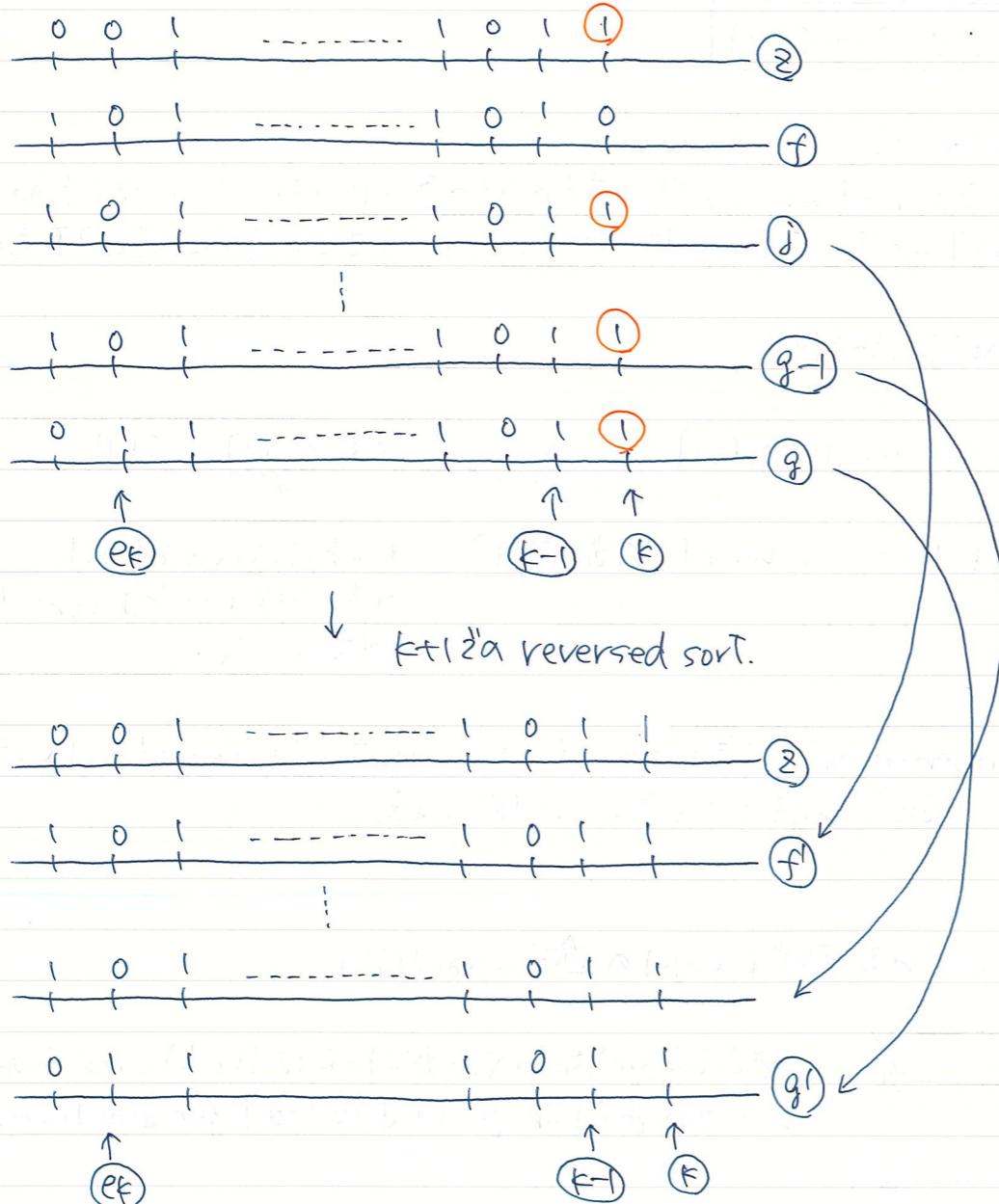


$(f_k, g_k)$  がまだ<sup>2</sup> ある<sup>2</sup> の要素数<sup>2</sup> は  $z[e_k, g_k]$  の match 値<sup>2</sup> である<sup>2</sup>。  
すなはち  $f_k$  の直前 or  $g_k$  の直後 (すなはち  $g_k$  の直前) である<sup>2</sup>。

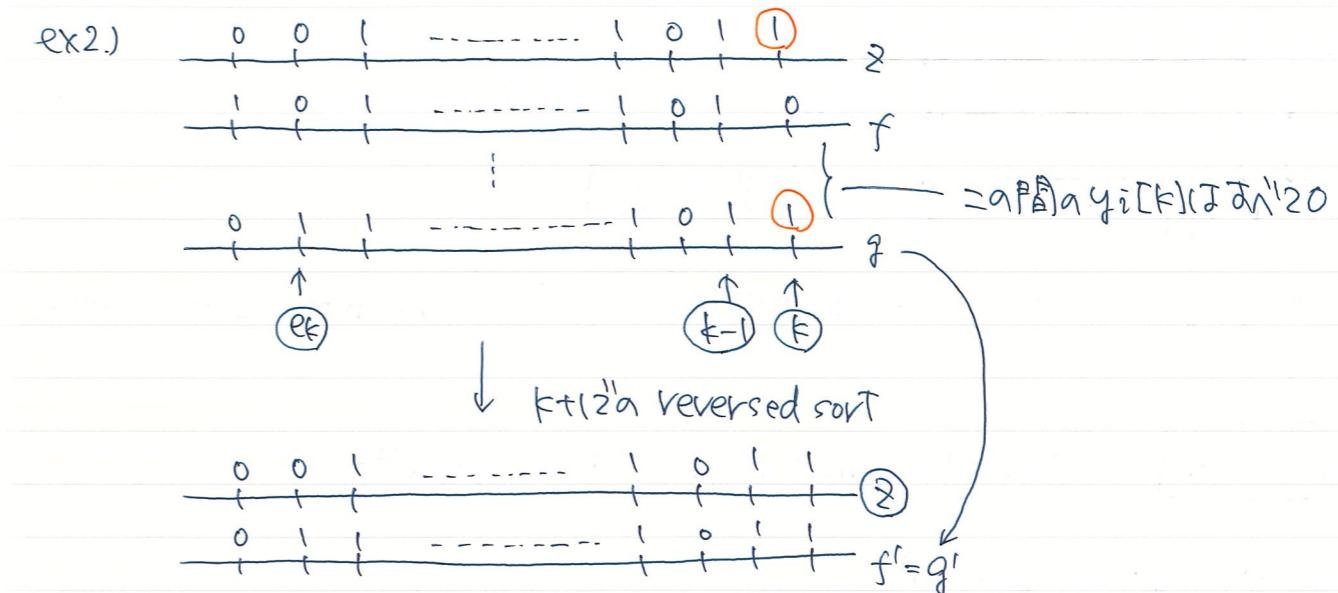
$\Rightarrow z' = w(f, z[k])$  とある。つまり  $f \geq z'$  のとき  $w(f, z[k]) = z[k]$  となるから、 $k+1$  における  $f$  は reversed sort において  $\oplus$  番目 (イニテル, ワク) である。

同様に  $g' = w(g, z[k])$  とある。つまり  $g \geq g'$  のとき  $w(g, z[k]) = z[k]$  となるから、 $k+1$  における  $g$  は reversed sort において  $\oplus$  番目 (イニテル, ワク) である。

ex1.)



$f' < g'$  の場合。この場合  $[e_k, k+1]$  が match であることは既に述べた。この中に存在する  $z$  の  $k+1$  における  $f$  は reversed sort における  $\oplus$  番目 (イニテル, ワク) である。この  $e_{k+1} = e_k$  である。

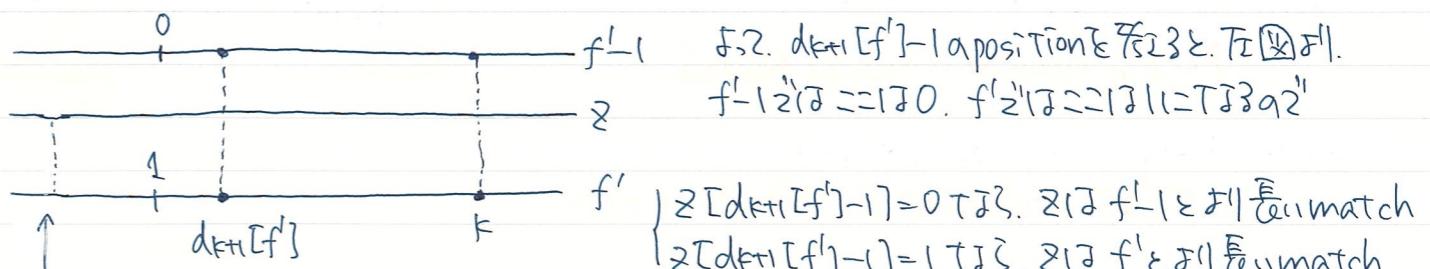


$f' = g'$  の場合。  
①  $[e_k, k+1]$  が match であることは既に述べた。この中に存在する  $z$  の  $k+1$  における  $f$  を探す必要がある。  
②  $[f_k, g_k]$  が  $k+1$  における set-maximal match ( $= T_j$ )

この場合  $k+1$  における reversed sort は  $f$  が  $z$  の  $k+1$  における  $f$  である。この  $f$  が  $k+1$  における reversed sort である。

$$y^{k+1} f'_1 < z < y^{k+1} f'$$

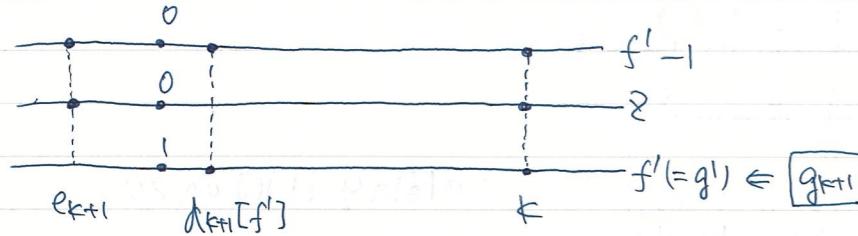
もし  $f'_1$  が  $z$  の  $k+1$  における  $f$  である。つまり  $e_{k+1} \leq d_{k+1}[f']$  が成立するはずである。



$e_k$  が  $d_{k+1}[f']$  である。  
 $T_j = T_i$  ( $e_k$  が  $f$  の  $k+1$  における set-maximal match)

もし  $f'_1$  が  $z$  の  $k+1$  における  $f$  である。つまり  $d_{k+1}[f'] - 1$  が position であると仮定する。  
 $f'_1$  が  $z$  の  $k+1$  における  $f$  である。つまり  $f'_1$  が  $z$  の  $k+1$  における set-maximal match

$f'_1$  が  $z$  の  $k+1$  における set-maximal match



この場合に  $f' = q' = g_{k+1}$  と(2)お $\exists$ ,  $f_{k+1} < f'$  の範囲で  $[e_{k+1}, k+1]$  が match が成立する  $\exists f_{k+1} \leq f' = q'$

**Algorithm 5: Code**  $k^{\text{th}}$  set-maximal match pos.  $k+1$  update $\exists$

$f' \leftarrow w(f_k, z[k])$ ,  $g' \leftarrow w(g_k, z[k])$

if  $f' < g'$  then  $e' \leftarrow e_k$ .

else  $i=f$ ;  $i < g$ ;  $i \leftarrow i+1$  do report match to  $a_k[i]$  from  $e_k$  to  $k$

( $f \leq i < g$  で  $T=f$  かつ  $i=e_f(k)$  かつ  $z[p][e_f(k)] \leq p$  "match" と報告)

$e' = d_{k+1}[f'] - 1$

if  $z[e'] = 0$  and  $f' > 0$  then

$f' \leftarrow g' - 1$   $\triangleq g'(i \text{ と } f' \text{ が } g' \text{ の前にある。上の図と同じ。})$

while  $z[e'-1] = y_{f'}^{k+1}[e'-1]$  do  $e' \leftarrow e' - 1$

$\triangleq z \in y_{f'}^{k+1} \text{ の match } \exists f' < e' \text{ 前に } e'$

while  $d_{k+1}[f'] \leq e'$  do  $f' \leftarrow f' - 1$

$\triangleq$  上で求めた  $f'$  の match  $\exists f' < e'$  上に  $e'$

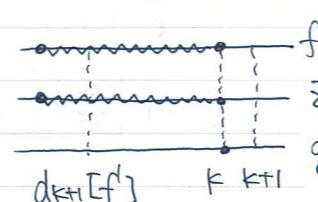
else: ( $z[e'] = 1$ )

$q' \leftarrow f' + 1$

while  $z[e'-1] = y_{f'}^{k+1}[e'-1]$  do  $e' \leftarrow e' - 1$

while  $q' < M$  and  $d_{k+1}[q'] \leq e'$  do  $q' \leftarrow q' + 1$

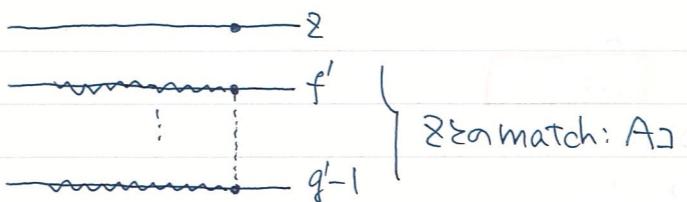
$f_{k+1} \leftarrow f'$ ,  $g_{k+1} \leftarrow q'$ ,  $e_{k+1} \leftarrow e'$   $\square$



この場合に(2)下の  $f' = T$  がニヤル開始

二つ: 内側のレポートのうち,  $f, g$  を含むメートル部分には,  $f' = q' = T$  は  $T=k$  の  $k=e_k(k+1)$  で起きた $\exists$ 。

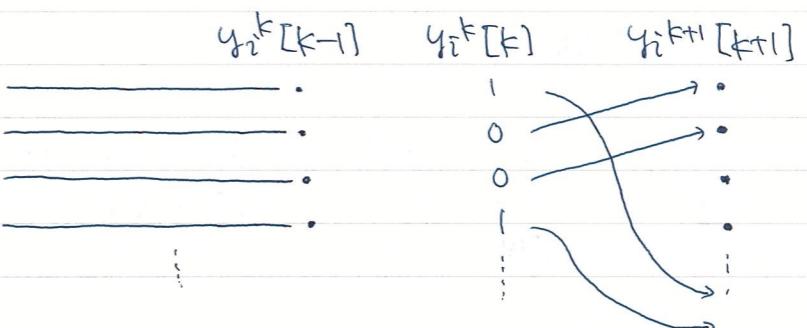
2. 合計 a set-maximal の個数は,  $T=k$  が MA 1 回に制限される



Xのコントラスト保存方法

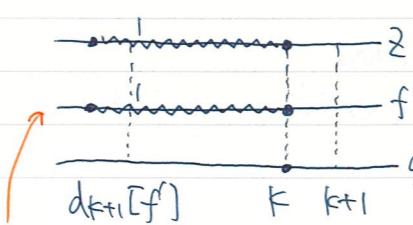
実際に情報が格納する場合は,  $y_i^k[k]$  の値が"重要"か"。これを位置  $k$  と組み合わせて保存 (2) だけ足す。

$y_i^k[k]$  を並べて  $\exists$  PBWT (Positional Burrows-Wheeler Transform) とする。



$(k+1-f') \text{ reversed sort } \exists$ . つまり"位置  $k+1-f'$  が 0 かつ "並び" に  $i$  のもの" 並びの 2": 自動的に対応関係が定義されるとの西引  $\exists$  復元される。

連鎖不均等  $\exists$  複数の  $\exists$  共通の PLT  $\exists$  と PBWT ( $= F$ ) 高圧縮が実現される。



Algorithm5 を使うには、 $w_f$  を使う、 $f$  の行先を決めるために  $U_f$  と  $U_f$  を求める必要がある。

FM-index と同じように (FM-index (= ヒツの面倒) と同じように)  $U_f$  と  $U_f$  を求める。

$a_k$	$U_k$	$b_k$	$U_k$
$a_k[0] (= \varnothing)$	$U_k[0] = 1$		$U_k[0] = 0$
$a_k[1]$	$U_k[1] = 1$	$a_k[1] (= \varnothing)$	$U_k[1] = 1$
$a_k[2]$	$U_k[2] = 1$	$a_k[2] (= \varnothing)$	$U_k[2] = 2$
$a_k[3]$	$U_k[3] = 2$		$U_k[3] = 2$
$a_k[4]$	$U_k[4] = 3$		$U_k[4] = 2$

FM-index と同じ要領で間を埋めていく。(この3つは  $U_k[1], U_k[2]$  が始まりに付かず本質的) (同じように)。

このとき,  $f = a_k[i']$  とする。  $f' = w(f, z[k])$  と  $f$  の行先を表すと、 $z[i']$  が  $f$  の行先となる。

$a_k[i]$	$U_k[i]$	$U_k[i]$
:	:	:
$a_k[i'-1]$	8	$a \neq l$ , $y_i^k[k] = 0$ で $3 = 2$ となる。
$a_k[i']$	9 <sup>a</sup>	$w(f, z[k] = 1)$ を $23 + 5$ , $b = 0$ で $23$ となる。
$a_k[i'+1]$	9	$7^b$

(Point) PBWT(子変異サブを固定した完全matchを求める場合の手順。  
(各サブが "biallelic" な場合にのみ適用可。)