

RLPBWT con bitvectors

Algorithm 1 Algoritmo per la costruzione di una colonna della RLPBWT con bitvectors

```

1: function BUILD(col, pref, div)
2:    $c \leftarrow 0$ ,  $u \leftarrow 0$ ,  $v \leftarrow 0$ ,  $u' \leftarrow 0$ ,  $v' \leftarrow 0$ ,  $curr_{lcs} \leftarrow 0$ ,  $tmp_{thr} \leftarrow 0$ ,  $tmp_{beg} \leftarrow 0$ 
3:    $start \leftarrow \top$ ,  $beg_{run} \leftarrow \top$ ,  $push_{zero} \leftarrow \perp$ ,  $push_{one} \leftarrow \perp$ 
4:   for every  $k \in [0, height)$  do
5:     if  $k = 0 \wedge col[pref[k]] = 1$  then
6:        $start \leftarrow \perp$ 
7:     if  $col[k] = 0$  then
8:        $c \leftarrow c + 1$ 
9:      $runs \leftarrow [0..0]$  ▷ sparse bitvector for runs of length  $height + 1$ 
10:     $thrs \leftarrow [0..0]$  ▷ sparse bitvector for thresholds of length  $height$ 
11:     $zeros \leftarrow [0..0]$  ▷ sparse bitvector for zeros of length  $c$ 
12:     $ones \leftarrow [0..0]$  ▷ sparse bitvector for ones of length  $height - c$ 
13:     $samples_{beg} \leftarrow []$ ,  $samples_{end} \leftarrow []$  ▷ couple of vectors for samples of length  $r$ 
14:    if  $start$  then
15:       $push_{one} \leftarrow \top$ 
16:    else
17:       $push_{zero} \leftarrow \top$ 
18:    for every  $k \in [0, height)$  do
19:      if  $beg_{run}$  then
20:         $u \leftarrow u'$ ,  $v \leftarrow v'$ ,  $tmp_{beg} \leftarrow pref[k]$ 
21:         $beg_{run} \leftarrow \perp$ 
22:      if  $col[pref[k]] = 1$  then
23:         $v' \leftarrow v' + 1$ 
24:      else
25:         $u' \leftarrow u' + 1$ 
26:      if  $k = 0 \vee col[pref[k]] \neq col[pref[k - 1]]$  then
27:         $curr_{lcs} \leftarrow div[k]$ ,  $tmp_{thr} \leftarrow k$ 
28:      if  $div[k] < curr_{lcs}$  then
29:         $curr_{lcs} \leftarrow div[k]$ ,  $tmp_{thr} \leftarrow k$ 
30:      if  $k = height - 1 \vee col[pref[k]] \neq col[pref[k + 1]]$  then
31:         $runs[k] \leftarrow 1$ 
32:        if  $k \neq height - 1 \wedge div[k + 1] < div[tmp_{thr}]$  then
33:           $thrs[k] \leftarrow 1$ 
34:        else
35:           $thrs[tmp_{thr}] \leftarrow 1$ 
36:           $push(samples_{beg}, tmp_{beg})$ 
37:           $push(samples_{end}, pref[k])$ 
38:          if  $push_{one}$  then
39:            if  $v \neq 0$  then
40:               $ones[k - 1] = 1$ 
41:               $swap(push_{zero}, push_{one})$ 
42:          else
43:            if  $u \neq 0$  then
44:               $zeros[k - 1] = 1$ 
45:               $swap(push_{zero}, push_{one})$ 
46:           $beg_{run} \leftarrow \top$ 
47:      if  $|zeros| \neq 0$  then
48:         $zeros[|zeros| - 1] \leftarrow 1$ 
49:      if  $|ones| \neq 0$  then
50:         $ones[|ones| - 1] \leftarrow 1$ 
51:      build rank/select for the four bitvectors
52:      return ( $start$ ,  $c$ ,  $runs$ ,  $zeros$ ,  $ones$ ,  $samples_{beg}$ ,  $samples_{end}$ ,  $div$ )

```

Algorithm 2 Algoritmo per estrazione simbolo da una run in una colonna

```
1: function GET_SYMBOL( $s, r$ )  $\triangleright s = \top$  iff column start with 0,  $r$  run index
2:   if  $s$  then
3:     if  $r \bmod 2 = 0$  then return 0 else return 1
4:   else
5:     if  $r \bmod 2 = 0$  then return 1 else return 0
```

Algorithm 3 Algoritmo per uvtrick

```
1: function UVTRICK( $k, i$ )  $\triangleright k$  is column index,  $i$  row index
2:   if  $i = 0$  then
3:     return (0, 0)
4:    $run \leftarrow rank_h^k(i)$ 
5:   if  $run = 0$  then
6:     if  $start^k$  then
7:       return ( $index, 0$ )
8:     else
9:       return (0,  $index$ )
10:  else if  $run = 1$  then
11:    if  $start^k$  then
12:      return ( $select_h^k(run) + 1, i - (select_h^k(run) + 1)$ )
13:    else
14:      return ( $i - (select_h^k(run) + 1), select_h^k(run) + 1$ )
15:  else
16:    if  $run \bmod 2 = 0$  then
17:       $pre_u \leftarrow select_u^k(\frac{run}{2}) + 1$ 
18:       $pre_v \leftarrow select_v^k(\frac{run}{2}) + 1$ 
19:       $offset \leftarrow i - (select_h^k(run) + 1)$ 
20:      if  $start^k$  then
21:        return ( $pre_u + offset, pre_v$ )
22:      else
23:        return ( $pre_u, pre_v + offset$ )
24:    else
25:       $run_u \leftarrow (\frac{run}{2}) + 1$ 
26:       $run_v \leftarrow \frac{run}{2}$ 
27:      if  $\neg start^k$  then
28:         $swap(run_u, run_v)$ 
29:       $pre_u \leftarrow select_u^k(run_u) + 1$ 
30:       $pre_v \leftarrow select_v^k(run_v) + 1$ 
31:       $offset \leftarrow i - (select_h^k(run) + 1)$ 
32:      if  $start^k$  then
33:        return ( $pre_u, pre_v + offset$ )
34:      else
35:        return ( $pre_u + offset, pre_v$ )
```

Algorithm 4 Algoritmo per lf-mapping

```
1: function LF( $k, i, s$ )  $\triangleright k$  is column index,  $i$  row index,  $s$  symbol
2:    $c \leftarrow rlpbwt[k].c$ 
3:    $(u, v) \leftarrow uvtrick(k, i)$ 
4:   if  $s = 0$  then
5:     return  $u$ 
6:   else
7:     return  $c + v$ 
```

Algorithm 5 Algoritmo per lf-mapping inverso

```
1: function REVERSE_LF( $k, i$ )  $\triangleright k$  is column index,  $i$  row index
2:   if  $k = 0$  then  $\triangleright$  by design
3:     return 0
4:    $k \leftarrow k - 1$ 
5:    $c \leftarrow rlpbwt[k].c$ 
6:   if  $i < c$  then
7:     if  $start^k$  then
8:        $run \leftarrow rank_u^k(i) \cdot 2$ 
9:     else
10:       $run \leftarrow rank_u^k(i) \cdot 2 + 1$ 
11:      $i_{run} \leftarrow 0$ 
12:     if  $run \neq 0$  then
13:        $i_{run} \leftarrow select_h^k(run) + 1$ 
14:      $(prev_0, \_) \leftarrow uvtrick(k, i_{run})$ 
15:     return  $i_{run} + (i - prev_0)$ 
16:   else
17:     if  $start^k$  then
18:        $run \leftarrow rank_v^k(i) \cdot 2 + 1$ 
19:     else
20:        $run \leftarrow rank_v^k(i) \cdot 2$ 
21:      $i_{run} \leftarrow 0$ 
22:     if  $run \neq 0$  then
23:        $i_{run} \leftarrow select_h^k(run) + 1$ 
24:      $(\_, prev_1) \leftarrow uvtrick(k, i_{run})$ 
25:     return  $i_{run} + (i - (c + prev_1))$ 
```

Algorithm 6 Algoritmo per match con aplotipo esterno con panel $width \times height$

```

1: function EXTERNAL_MATCHES( $z$ ) ▷ assuming  $|z| = rlpbwt.width$ 
2:    $f \leftarrow 0, f_{run} \leftarrow 0, f' \leftarrow 0$ 
3:    $g \leftarrow 0, g_{run} \leftarrow 0, g' \leftarrow 0$ 
4:    $e \leftarrow 0, l \leftarrow 0$ 
5:   for every  $k \in [0, |z|)$  do
6:      $f_{run} \leftarrow rank_h^k(f), g_{run} \leftarrow rank_h^k(g)$ 
7:      $f' \leftarrow lf(k, f, z[k]), g' \leftarrow lf(k, g, z[k])$ 
8:      $l \leftarrow g - f$ 
9:     if  $f' < g'$  then
10:       $f \leftarrow f', g \leftarrow g'$ 
11:     else
12:       if  $k \neq 0$  then
13:         report matches in  $[e, k - 1]$  with  $l$  haplotypes
14:       if  $f' = |lcp^{k+1}|$  then
15:          $e \leftarrow k + 1$ 
16:       else
17:          $e \leftarrow lcp^{k+1}[f']$ 
18:       if  $(z[e] = 0 \wedge f' > 0) \vee f' = height$  then
19:          $f' \leftarrow g' - 1$ 
20:         if  $e \geq 1$  then
21:            $f_{rev} \leftarrow f', k' \leftarrow k + 1$ 
22:           while  $k' \neq e - 1$  do
23:              $f_{rev} \leftarrow reverse\_lf(k', f_{rev}), k' \leftarrow k' - 1$ 
24:            $run \leftarrow rank_h^{k'}(f_{rev}), symb \leftarrow get\_symbol(start^{k'}, run)$ 
25:           while  $e > 0 \wedge z[e - 1] = symb$  do
26:              $f_{rev} \leftarrow reverse\_lf(e, f_{rev})$ 
27:              $run \leftarrow rank_h^{e-1}(f_{rev})$ 
28:              $symb \leftarrow get\_symbol(start^{e-1}, run)$ 
29:           while  $f' > 0 \wedge (k + 1) - lcp^{k+1}[f] \leq e$  do  $e \leftarrow e - 1$ 
30:            $f \leftarrow f', g \leftarrow g'$ 
31:         else
32:            $g' \leftarrow f' - 1$ 
33:           if  $e \geq 1$  then
34:              $f_{rev} \leftarrow f', k' \leftarrow k + 1$ 
35:             while  $k' \neq e - 1$  do
36:                $f_{rev} \leftarrow reverse\_lf(k', f_{rev}), k' \leftarrow k' - 1$ 
37:              $run \leftarrow rank_h^{k'}(f_{rev}), symb \leftarrow get\_symbol(start^{k'}, run)$ 
38:             while  $e > 0 \wedge z[e - 1] = symb$  do
39:                $f_{rev} \leftarrow reverse\_lf(e, f_{rev})$ 
40:                $run \leftarrow rank_h^{e-1}(f_{rev})$ 
41:                $symb \leftarrow get\_symbol(start^{e-1}, run)$ 
42:             while  $e < height \wedge (k + 1) - lcp^{k+1}[e] \leq e$  do  $e \leftarrow e + 1$ 
43:              $f \leftarrow f', g \leftarrow g'$ 
44:       if  $f < g$  then
45:          $l \leftarrow g - f$ 
46:       report matches in  $[e, |z| - 1]$  with  $l$  haplotypes

```

Algorithm 7 Algoritmo per match con matching-statistics (MS) e thresholds

```

1: function MATCHES_MS( $z$ )
2:    $ms_{row} \leftarrow [0..0]$ ,  $ms_{len} \leftarrow [0..0]$  ▷ ms vectors with row and len of length  $|z|$ 
3:    $curr_{row} \leftarrow rlpbwt[0].samples_{end}[|rlpbwt[0].samples_{end}| - 1]$ 
4:    $curr_{index} \leftarrow curr_{row}$ 
5:    $curr_{run} \leftarrow rank_h^0(curr_{index})$ 
6:    $symb \leftarrow get\_symbol(start^0, curr_{run})$  ▷ build matching statistics row
7:   for every  $k \in [0, |z|)$  do
8:     if  $z[i] = symb$  then
9:        $ms_{row}[k] \leftarrow curr_{row}$ 
10:      if  $k \neq |z| - 1$  then
11:         $(curr_{index}, curr_{run}, symb) \leftarrow UPDATE(k, curr_{index}, z)$ 
12:      else
13:         $curr_{thr} \leftarrow rank_t^k(curr_{index})$ 
14:         $force_{down} \leftarrow \top$  iff we are over a threshold not at the end of a run
15:         $force_{down} \leftarrow \top$  iff we are over a threshold at the end of a run and DOWN function is  $\top$ 
16:        if  $|samples_{beg}^k| = 1$  then
17:           $ms_{row}[k] \leftarrow height$ 
18:          if  $k \neq |z| - 1$  then
19:             $curr_{row} \leftarrow rlpbwt[k+1].samples_{end}[|rlpbwt[k+1].samples_{end}| - 1]$ 
20:             $curr_{index} \leftarrow height - 1$ 
21:             $curr_{run} \leftarrow rank_h^{k+1}(curr_{index})$ 
22:             $symb \leftarrow get\_symbol(start^{k+1}, curr_{run})$ 
23:          else if  $(curr_{run} \neq 0 \wedge curr_{run} = curr_{thr} \wedge \neg down) \vee curr_{run} = |samples_{beg}^k| - 1$  then
24:             $curr_{index} \leftarrow select_h^k(curr_{run})$ 
25:             $curr_{row} \leftarrow samples_{end}^k[curr_{run} - 1]$ 
26:             $ms_{row}[k] \leftarrow curr_{row}$ 
27:            if  $k \neq |z| - 1$  then
28:               $(curr_{index}, curr_{run}, symb) \leftarrow UPDATE(k, curr_{index}, z)$ 
29:            else
30:               $curr_{index} \leftarrow select_h^k(curr_{run} + 1) + 1$ 
31:               $curr_{row} \leftarrow samples_{beg}^k[curr_{run} + 1]$ 
32:               $ms_{row}[k] \leftarrow curr_{row}$ 
33:              if  $k \neq |z| - 1$  then
34:                 $(curr_{index}, curr_{run}, symb) \leftarrow UPDATE(k, curr_{index}, z)$  ▷ build matching statistics len
35:      for every  $k \in [0, |ms_{row}|)$  do
36:        if  $ms_{row}[k] = height$  then
37:           $ms_{len}[k] \leftarrow 0$ 
38:        else if  $k \neq 0 \wedge ms_{row}[i] = ms_{row}[i-1] \wedge ms_{len}[i-1] \neq 0$  then
39:           $ms_{len}[i] \leftarrow ms_{len}[i-1] + 1$ 
40:        else ▷  $ra$  is a data structure for random access over the originale panel
41:           $tmp_{index} \leftarrow i$ ,  $tmp_{len} \leftarrow 0$ 
42:          while  $tmp_{index} \geq 0 \wedge z[tmp_{index}] = ra(ms_{row}[k], tmp_{index})$  do
43:             $tmp_{index} \leftarrow tmp_{index} - 1$ ,  $tmp_{len} \leftarrow tmp_{len} + 1$ 
44:           $ms_{len}[k] \leftarrow tmp_{len}$ 
45:      for every  $k \in [0, |ms_{row}|)$  do ▷ build matching statistics matches
46:        if  $(ms_{len}[k] > 1 \wedge ms_{len}[k] \geq ms_{len}[k+1]) \vee (k = |z| - 1 \wedge ms_{len}[k] \neq 0)$  then
47:          report match ending in  $k$ , with length  $ms_{len}[k]$ , with at least row  $ms_{row}[k]$ 
in case extend the matches

```

function DOWN($pos, prev, next$)

using LCE queries or random access check the longest common prefix between
 pos and $prev$ and between pos and $next$
if the latter is greater or equal return \top , else \perp

Algorithm 8 Algoritmo per match con matching-statistics (MS) e LCE

```

1: function MATCHES_MS_LCE( $z$ )
2:    $ms_{row} \leftarrow [0..0]$ ,  $ms_{len} \leftarrow [0..0]$  ▷ ms vectors with row and len of length  $|z|$ 
3:    $curr_{row} \leftarrow rlpbwt[0].samples_{end}[rlpbwt[0].samples_{end} - 1]$ 
4:    $curr_{index} \leftarrow curr_{row}$ ,  $curr_{run} \leftarrow rank_h^0(curr_{index})$ 
5:    $symb \leftarrow get\_symbol(start^0, curr_{run})$  ▷ build matching statistics row
6:   for every  $k \in [0, |z|)$  do
7:     if  $z[i] = symb$  then
8:        $ms_{row}[k] \leftarrow curr_{row}$ 
9:       if  $k = 0$  then
10:         $ms_{len}[k] \leftarrow 1$ 
11:       else
12:         $ms_{len}[k] \leftarrow ms_{len}[k - 1] + 1$ 
13:       if  $k \neq |z| - 1$  then
14:         $(curr_{index}, curr_{run}, symb) \leftarrow UPDATE(k, curr_{index}, z)$ 
15:       else
16:        if  $|samples_{beg}^k| = 1$  then
17:           $ms_{row}[k] \leftarrow height$ 
18:           $ms_{len}[k] \leftarrow 0$ 
19:          if  $k \neq |z| - 1$  then
20:             $curr_{row} \leftarrow rlpbwt[k + 1].samples_{end}[rlpbwt[k + 1].samples_{end} - 1]$ 
21:             $curr_{index} \leftarrow height - 1$ 
22:             $curr_{run} \leftarrow rank_h^{k+1}(curr_{index})$ 
23:             $symb \leftarrow get\_symbol(start^{k+1}, curr_{run})$ 
24:          else
25:            if  $curr_{run} = |samples_{beg}^k| - 1$  then
26:               $curr_{index} \leftarrow select_h^k(curr_{run})$ ,  $prev_{row} \leftarrow samples_{end}^k[curr_{run} - 1]$ 
27:               $lce \leftarrow LCE(k, curr_{row}, prev_{row})$ 
28:               $ms_{row}[k] \leftarrow prev_{row}$ ,  $curr_{row} \leftarrow prev_{row}$ 
29:              if  $k = 0$  then
30:                 $ms_{len}[k] \leftarrow 1$ 
31:              else
32:                 $ms_{len}[k] \leftarrow \min(ms_{len}[k - 1], lce_{len}) + 1$ 
33:              if  $k \neq |z| - 1$  then
34:                 $(curr_{index}, curr_{run}, symb) \leftarrow UPDATE(k, curr_{index}, z)$ 
35:            else if  $curr_{run} = 0$  then
36:               $curr_{index} \leftarrow select_h^k(curr_{run} + 1) + 1$ ,  $next_{row} \leftarrow samples_{beg}^k[curr_{run} + 1]$ 
37:               $lce \leftarrow LCE(k, curr_{row}, next_{row})$ 
38:               $ms_{row}[k] \leftarrow next_{row}$ ,  $curr_{row} \leftarrow next_{row}$ 
39:              if  $k = 0$  then
40:                 $ms_{len}[k] \leftarrow 1$ 
41:              else
42:                 $ms_{len}[k] \leftarrow \min(ms_{len}[k - 1], lce_{len}) + 1$ 
43:              if  $k \neq |z| - 1$  then
44:                 $(curr_{index}, curr_{run}, symb) \leftarrow UPDATE(k, curr_{index}, z)$ 
45:            else
46:               $prev_{row} \leftarrow samples_{end}^k[curr_{run} - 1]$ ,  $next_{row} \leftarrow samples_{beg}^k[curr_{run} + 1]$ 
47:               $lce \leftarrow \max_{len}(LCE(k, curr_{row}, prev_{row}), LCE(k, curr_{row}, next_{row}))$ 
48:               $curr_{row} \leftarrow lce_{row}$ 
49:               $ms_{row}[k] \leftarrow curr_{row}$ 
50:              if  $k = 0$  then
51:                 $ms_{len}[k] \leftarrow 1$ 
52:              else
53:                 $ms_{len}[k] \leftarrow \min(ms_{len}[k - 1], lce_{len}) + 1$ 
54:              if  $k \neq |z| - 1$  then
55:                 $(curr_{index}, curr_{run}, symb) \leftarrow UPDATE(k, curr_{index}, z)$ 
56:          for every  $k \in [0, |ms_{row}|)$  do ▷ build matching statistics matches
57:            if  $(ms_{len}[k] > 1 \wedge ms_{len}[k] \geq ms_{len}[k + 1]) \vee (k = |z| - 1 \wedge ms_{len}[k] \neq 0)$  then
58:              report match ending in  $k$ , with length  $ms_{len}[k]$ , with at least row  $ms_{row}[k]$ 
in case extend the matches

```

Algorithm 9 Algoritmo per l'update usando le matching statistics

```

1: function UPDATE( $k, curr\_index, z$ )
2:    $curr\_index \leftarrow lf(k, curr\_index, z[k])$ 
3:    $curr\_run \leftarrow rank_h^{k+1}(curr\_index)$ 
4:    $symb \leftarrow get\_symbol(start^{k+1}, curr\_run)$ 
5:   return ( $curr\_index, curr\_run, symb$ )

```

Algorithm 10 Algoritmo per la costruzione della struttura per φ e φ^{-1}

```

1: function BUILD_PHI( $cols, panel, prefix$ ) ▷  $prefix$  is the last prefix array
2:    $\varphi \leftarrow [[0..0]..[0..0]], \varphi^{-1} \leftarrow [[0..0]..[0..0]]$  ▷ sparse bit vector panels for  $\varphi$  and  $\varphi^{-1}$ 
3:    $\varphi_{supp} = [], \varphi_{supp}^{-1} = []$  ▷ vectors for  $\varphi$  and  $\varphi^{-1}$  row values
4:   for every  $k \in [0, |cols|)$  do
5:     for every  $i \in [0, |samples_{beg}|)$  do
6:        $\varphi[sample_{beg}^k[i]][k] \leftarrow 1$ 
7:       if  $i = 0$  then
8:          $push(\varphi_{supp}[sample_{beg}^k[i]], panel_{height})$ 
9:       else
10:         $push(\varphi_{supp}[sample_{beg}^k[i]], sample_{end}^k[i - 1])$ 
11:       $\varphi^{-1}[sample_{end}^k[i]][k] \leftarrow 1$ 
12:      if  $i = |sample_{beg}^k| - 1$  then
13:         $push(\varphi_{supp}^{-1}[sample_{end}^k[i]], panel_{height})$ 
14:      else
15:         $push(\varphi_{supp}^{-1}[sample_{end}^k[i]], sample_{beg}^k[i + 1])$ 
16:    for every  $k \in [0, |prefix|)$  do
17:      if  $\varphi[k][|\varphi[k]| - 1] = 0$  then
18:         $\varphi[k][|\varphi[k]| - 1] \leftarrow 1$ 
19:        if  $k = 0$  then
20:           $push(\varphi_{supp}[prefix^k], panel_{height})$ 
21:        else
22:           $push(\varphi_{supp}[prefix^k], prefix^k[i - 1])$ 
23:        if  $\varphi^{-1}[k][|\varphi[k]| - 1] = 0$  then
24:           $\varphi^{-1}[k][|\varphi[k]| - 1] \leftarrow 1$ 
25:          if  $k = |prefix| - 1$  then
26:             $push(\varphi_{supp}^{-1}[prefix^k], panel_{height})$ 
27:          else
28:             $push(\varphi_{supp}^{-1}[prefix^k], prefix^k[i + 1])$ 
29:    build rank/select for every sparse bitvector in  $\varphi$  and  $\varphi^{-1}$ 

```

Algorithm 11 Algoritmi per le query a φ e φ^{-1}

```
1: function  $\varphi(prefix_{value}, col)$ 
2:    $res \leftarrow \varphi_{supp}^{prefix_{value}}[rank_{\varphi}^{prefix_{value}}(col)]$ 
3:   if  $res = panel_{height}$  then
4:     return  $null$ 
5:   else
6:     return  $res$ 
1: function  $\varphi^{-1}(prefix_{value}, col)$ 
2:    $res \leftarrow \varphi_{supp}^{-1, prefix_{value}}[rank_{\varphi^{-1}}^{prefix_{value}}(col)]$ 
3:   if  $res = panel_{height}$  then
4:     return  $null$ 
5:   else
6:     return  $res$ 
```

Algorithm 12 Algoritmo per estendere un match in col usando φ , φ^{-1} e MS

```
1: function EXTEND_MATCHES( $col, row, len$ )
2:    $check_{down} \leftarrow \top$ ,  $check_{up} \leftarrow \top$ 
3:   while  $check_{down}$  do
4:      $down_{row} \leftarrow \varphi^{-1}(row, col)$ 
5:     if  $lce\_bounded(col, row, down_{row}, len)$  then
6:        $push(haplos, down_{row})$ 
7:        $row \leftarrow down_{row}$ 
8:     else
9:        $check_{down} \leftarrow \perp$ 
10:  while  $up_{down}$  do
11:     $up_{row} \leftarrow \varphi(row, col)$ 
12:    if  $lce\_bounded(col, row, up_{row}, len)$  then
13:       $push(haplos, up_{row})$ 
14:       $row \leftarrow up_{row}$ 
15:    else
16:       $check_{up} \leftarrow \perp$ 
17:  return  $haplos$ 
```
