Algoritmi 4.4.3 *LCP-taulukon muodostaminen*

Syöte: Merkkijono $S = S[0 \ldots n)$, loppuosataulukko $SA[0 \ldots n]$
ja sen käänteistaulukko $SA^{-1}[0 \ldots n]$.
Tuloste: LCP-taulukko $LCP[0 \ldots n]$.

```
(1) ℓ := 0;
(2) for i := 0 to n − 1 do
(3)     k := SA⁻¹[i];                 (* i = SA[k] *)
(4)     if k = n then LCP[k] := 0;
(5)     else
(6)         j := SA[k + 1];
(7)         ℓ := ℓ + lcp(S_{i+ℓ}, S_{j+ℓ});
(8)         LCP[k] := ℓ;              (* LCP[k] := lcp(S_i, S_j) *)
(9)     if ℓ > 0 then ℓ := ℓ − 1;
```

Lause 4.4.4 *Annettuna merkkijono $S[0 \ldots n)$ ja sen loppuosataulukko $SA[0 \ldots n]$, vastaava LCP-taulukko $LCP[0 \ldots n]$ voidaan laskea ajassa $\mathcal{O}(n)$.*

Todistus.

- Algoritmi toimii selvästi lineaarisessa ajassa lukuunottamatta rivin 7 lcp-arvon laskemista.

- Jokainen ylimääräinen merkkivertailu rivillä 7 kasvattaa ℓ:ää.

- Koska ℓ pienenee enintään yhdellä per kierros, eikä koskaan kasva suuremmaksi kuin n, rivillä 7 tehdään yhteensä $\mathcal{O}(n)$ merkkivertailua.

□

4.5 Loppuosataulukon muodostaminen

S. J. Puglisi, W. F. Smyth & A. Turpin. A taxonomy of suffix array construction algorithms, *ACM Computing Surveys*, 2007.

- Tarkastellaan nyt algoritmeja loppuosataulukon muodostamiseen.

- Tehtävänä on siis järjestää merkkijonon $S[0, n)$ loppuosat aakkosjärjestykseen.

- Normaali merkkijonojen järjestämisalgoritmi vie pahimmassa tapauksessa ajan $\Omega(n^2)$.
- Loppuosataulukko voidaan muodostaa laskemalla ensin loppuosapuu ja siitä loppuosataulukko. Tämä toimii myös toiseen suuntaan: loppuosapuu on helppo muodostaa lineaarisessa ajassa loppuosataulukosta ja LCP-taulukosta.
- Esitetään nyt tehokkaita algoritmeja, joilla loppuosataulukko voidaan muodostaa suoraan ilman loppuosapuuta.

4.5.1 Karp-Miller-Rosenberg nimentätekniikka

R.M. Karp & R.E. Miller & A.L. Rosenberg: Rapid Identification of Repeated Patterns in Strings Trees and Arrays. *Proceedings of the 4th Symposium on Theory of Computing*, sivut 125–136, 1972.

- Merkitään $S_i^\ell = S[i \ldots \min\{i+\ell, n\})$.
- Muodostetaan taulukot $Name_\ell[0,n]$, jotka sisältävät kokonaislukuja väliltä $[0,n]$ ja toteuttavat kaikilla $i,j \in [0,n]$

$$Name_\ell[i] \leq Name_\ell[j] \Longleftrightarrow S_i^\ell \leq S_i^\ell$$

- Taulukko $Name_\ell[0,n]$ antaa siis ℓ:n pituisten osajonojen ja ℓ:ää lyhyempien loppusosien nimennän, joka säilyttää jonojen keskinäisen järjestyksen.
- Erityisesti, kun $\ell \geq n$, saadaan kaikkien loppuosien nimentä. Siis $Name_\ell[i] = SA^{-1}[i]$, mistä saadaan helposti loppuosataulukko.
- Yllä oleva nimennän määritelmä ei ole yksikäsitteinen, mutta nimeksi voi valita

$$Name_\ell[i] = |\{j \in [0,n] \mid S_j^\ell < S_i^\ell\}|$$

- Taulukko $Name_1[0,n]$ voidaan laskea järjestämällä yhden pituiset jonot, siis jonon S merkit, ajassa $\mathcal{O}(n \log n)$.
- Taulukko $Name_{2\ell}[0,n]$ voidaan muodostaa taulukosta $Name_\ell[0,n]$ järjestämällä parit $(Name_\ell[i], Name_\ell[i+\ell])$. Tämä voidaan tehdä $\mathcal{O}(n)$ ajassa käyttämällä lopusta alkuun kantalukujärjestämistä.

- Loppuosataulukko (tai sen käänteistaulukko) saadaan toistamalla kahdentamisaskelta $\lceil \log n \rceil$ kertaa eli laskemalla $Name_1$, $Name_2$, $Name_4$, $Name_8$, ...

Esimerkki 4.5.1 *Käänteisen loppuosataulukon laskenta jonolle* ***banana****.*

$Name_1$:

4	b
1	a
5	n
1	a
5	n
1	a
0	

$Name_2$:

4	ba
2	an
5	na
2	an
5	na
1	a
0	

$Name_4$:

4	bana
3	anan
6	nana
2	ana
5	na
1	a
0	

$Name_8 = SA^{-1}$:

4	banana
3	anana
6	nana
2	ana
5	na
1	a
0	

Lause 4.5.2 *Merkkijonon $S[0, n]$ loppuosataulukko voidaan laskea ajassa $\mathcal{O}(n \log n)$.*

- Karp-Miller-Rosenberg nimentätekniikkaa voidaan käyttää myös muihin ongelmiin.

- Loppuosataulukon muodostamiseen sitä sovelsivat ensin Manber ja Myers.

- Käytännössä parhaan version tästä algoritmista ovat esittäneet Larsson ja Sadakane, joiden algoritmi sisältää joitakin merkkijonopikajärjestämisen piirteitä.

4.5.2 Osarekursio

J. Kärkkäinen & P. Sanders & S. Burkhardt: Linear Work Suffix Array Construction. *Journal of the ACM* 53(6):918–936, 2006.

- Esitetään nyt algoritmi, joka laskee loppuosataulukon ajassa $\mathcal{O}(n)$, kun aakkosto on $[1, n]$.

- Jos aakkosto on joku muu kuin $[1, n]$, voidaan kukin merkki $S[i]$ korvata nimellä $Name_1[i]$, jolloin päästään aakkostoon $[1, n]$ muuttamatta loppuosien keskinäistä järjestystä.

- Siten millä tahansa aakkostolla jonon S loppuosat voidaan järjestää samassa ajassa kuin S:n merkit.

- Perusrakenne on seuraava osarekursio:

  0. Jaa loppuosat kahteen ryhmään S_A ja S_B.
  1. Järjestä S_A. Tämä tapahtuu palauttamalla ongelma loppuosataulukon muodotukseen jonolle, jonka pituus on $|A|$.
  2. Järjestä S_B käyttäen S_A:n järjestystä.
  3. Lomita S_A ja S_B.

- Rekursio poislukien kaikki pystytään tekemään lineaarisessa ajassa.

- Lisäksi pätee $|A| \leq 2n/3$. Siten aikavaatimus on $T(n) = \mathcal{O}(n) + T(2n/3) = \mathcal{O}(n)$.

- Olennaista on, että S_B:tä ei järjestetä rekursiivisesti.

- Samankaltainen osarekursiivinen rakenne esiintyy kaikissa loppuosataulukon ja -puun muodostusalgoritmeissa, jotka toimivat lineaarisessa ajassa aakkostolla $[1, n]$.

- Syötteenä on siis jono $S[0, n)$ aakkostossa $[1, n]$. Lisäksi oletetaan, että $S[n, n + 3) = 000$.

Esimerkki 4.5.3

$$S[0,n) = \begin{array}{cccccccccccc} \scriptstyle 0 & \scriptstyle 1 & \scriptstyle 2 & \scriptstyle 3 & \scriptstyle 4 & \scriptstyle 5 & \scriptstyle 6 & \scriptstyle 7 & \scriptstyle 8 & \scriptstyle 9 & \scriptstyle 10 & \scriptstyle 11 \\ \mathtt{y} & \mathtt{a} & \mathtt{b} & \mathtt{b} & \mathtt{a} & \mathtt{d} & \mathtt{a} & \mathtt{b} & \mathtt{b} & \mathtt{a} & \mathtt{d} & \mathtt{o} \end{array}$$

Askel 0: Muodosta S_A.

- Määritellään kaikilla $k = 0,1,2$, $C_k = \{i \in [0,n] \mid i \bmod 3 = k\}$. Olkoon $A = C_1 \cup C_2$ ja $B = C_0$.

Esimerkki 4.5.4 $B = C_0 = \{0,3,6,9,12\}$, $C_1 = \{1,4,7,10\}$, $C_2 = \{2,5,8,11\}$, *eli* $A = \{1,4,7,10,2,5,8,11\}$.

Askel 1: Järjestä S_A.

- Muodosta jonot

$$R_k = \Big[S[k \dots k+3)\Big]\Big[S[k+3 \dots k+6)\Big] \dots \Big[S[\max C_k \dots \max C_k + 3)\Big]$$

$k = 1,2$, joiden merkit ovat kolmen merkin pituisia S:n osajonoja.

- Olkoon R jonojen R_1 ja R_2 katenaatio. Sen loppuosat vastaavat S_A:n loppuosia.

Esimerkki 4.5.5 $R = [abb][ada][bba][do0][bba][dab][bad][o00]$.

- Korvataan R:n merkit KMR-tekniikan mukaisilla nimillä. Tämä vaatii R:n merkkien järjestämistä, mikä voidaan tehdä lopusta alkuun kantalukujärjestämisellä lineaarisessa ajassa. Olkoon R' muunnettu jono.
- Muodostetaan rekursiivisella kutsulla R':n loppuosataulukko, mistä saadaan S_A:n järjestys. Rekursiota ei tarvita, jos kaikki R':n merkit ovat erilaiset.

Esimerkki 4.5.6 $R' = (1,2,4,6,4,5,3,7)$ *ja*
$SA_{R'} = (8,0,1,6,4,2,5,3,7)$.

- Olkoon $rank(i)$, $i \in A$, loppuosan S_i järjestysluku joukossa S_A. Lisäksi määritellään $rank(n+1) = rank(n+2) = 0$. Kun $i \in C_0$, $rank(i) = \perp$.

Esimerkki 4.5.7

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$rank(S_i)$	$\perp$	1	4	$\perp$	2	6	$\perp$	5	3	$\perp$	7	8	$\perp$	0	0

Askel 2: Järjestä S_B.

- Muodostetaan jokaiselle $S_i \in S_B$ pari $(t_i, rank(S_{i+1}))$. Huomaa, että $rank(S_{i+1})$ on aina määritelty.
- Selvästi kaikilla $i, j \in B$,

$$S_i \leq S_j \iff (S[i], rank(S_{i+1})) \leq (S[j], rank(S_{j+1})) \ .$$

- Parit $(S[i], rank(S_{i+1}))$ järjestetään lopusta alkuun kantalukujärjestämisellä.

Esimerkki 4.5.8 *$S_{12} < S_6 < S_9 < S_3 < S_0$ koska $(\mathtt{0}, 0) < (\mathtt{a}, 5) < (\mathtt{a}, 7) < (\mathtt{b}, 2) < (\mathtt{y}, 1)$.*

Askel 3: Lomita.

- Lomitetaan S_A ja S_B normaalilla vertailuihin perustuvalla lomitusalgoritmilla.
- Kahden loppuosan $S_i \in S_A$ ja $S_j \in S_B$ vertailussa erotetaan kaksi tapausta:

$$\begin{array}{ll} i \in C_1 : & S_i \leq S_j \iff (S[i], rank(i+1)) \leq (S[j], rank(j+1)) \\ i \in C_2 : & S_i \leq S_j \iff (S[i], S[i+1], rank(i+2)) \leq (S[j], S[j+1], rank(j+2)) \end{array}$$

Esimerkki 4.5.9 *$S_1 < S_6$ koska $(\mathtt{a}, 4) < (\mathtt{a}, 5)$ ja $S_3 < S_8$ koska $(\mathtt{b}, \mathtt{a}, 6) < (\mathtt{b}, \mathtt{a}, 7)$.*

Toteutus

- Alla on algoritmin täydellinen C++-kielinen toteutus.
- Toteutus ei ole tehokkain mahdollinen; pyrkimys on lyhyyteen (50 riviä poislukien tyhjät rivit, kommentit, ja rivit, joilla on vain aaltosulku).

```
inline bool leq(int a1, int a2,   int b1, int b2) // lexicographic order
{ return(a1 < b1 || a1 == b1 && a2 <= b2); }                // for pairs
inline bool leq(int a1, int a2, int a3,   int b1, int b2, int b3)
{ return(a1 < b1 || a1 == b1 && leq(a2,a3, b2,b3)); }     // and triples
```

```
// stably sort a[0..n-1] to b[0..n-1] with keys in 0..K from r
static void radixPass(int* a, int* b, int* r, int n, int K)
{ // count occurrences
  int* c = new int[K + 1];                          // counter array
  for (int i = 0;  i <= K;  i++) c[i] = 0;         // reset counters
  for (int i = 0;  i < n;  i++) c[r[a[i]]]++;   // count occurrences
  for (int i = 0, sum = 0;  i <= K;  i++)   // exclusive prefix sums
  {  int t = c[i];  c[i] = sum;  sum += t; }
  for (int i = 0;  i < n;  i++) b[c[r[a[i]]]++] = a[i];      // sort
  delete [] c;
}

// find the suffix array SA of T[0..n-1] in {1..K}^n
// require T[n]=T[n+1]=T[n+2]=0, n>=2
void suffixArray(int* T, int* SA, int n, int K) {
  int n0=(n+2)/3, n1=(n+1)/3, n2=n/3, n02=n0+n2;
  int* R  = new int[n02 + 3];  R[n02]= R[n02+1]= R[n02+2]=0;
  int* SA12 = new int[n02 + 3]; SA12[n02]=SA12[n02+1]=SA12[n02+2]=0;
  int* R0   = new int[n0];
  int* SA0  = new int[n0];

  //******* Step 0: Construct sample ********
  // generate positions of mod 1 and mod 2 suffixes
  // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1
  for (int i=0, j=0;  i < n+(n0-n1);  i++) if (i%3 != 0) R[j++] = i;

  //******* Step 1: Sort sample suffixes ********
  // lsb radix sort the mod 1 and mod 2 triples
  radixPass(R , SA12, T+2, n02, K);
  radixPass(SA12, R , T+1, n02, K);
  radixPass(R , SA12, T  , n02, K);

  // find lexicographic names of triples and
  // write them to correct places in R
  int name = 0, c0 = -1, c1 = -1, c2 = -1;
  for (int i = 0;  i < n02;  i++) {
    if (T[SA12[i]] != c0 || T[SA12[i]+1] != c1 || T[SA12[i]+2] != c2)
    {  name++;  c0 = T[SA12[i]];  c1 = T[SA12[i]+1];  c2 = T[SA12[i]+2]; }
    if (SA12[i] % 3 == 1) { R[SA12[i]/3]      = name; } // write to R1
    else                  { R[SA12[i]/3 + n0] = name; } // write to R2
  }

  // recurse if names are not yet unique
  if (name < n02) {
    suffixArray(R, SA12, n02, name);
    // store unique names in R using the suffix array
    for (int i = 0;  i < n02;  i++) R[SA12[i]] = i + 1;
  } else // generate the suffix array of R directly
    for (int i = 0;  i < n02;  i++) SA12[R[i] - 1] = i;
```

```
  //******* Step 2: Sort nonsample suffixes ********
  // stably sort the mod 0 suffixes from SA12 by their first character
  for (int i=0, j=0;  i < n02;  i++) if (SA12[i] < n0) R0[j++] = 3*SA12[i];
  radixPass(R0, SA0, T, n0, K);

  //******* Step 3: Merge ********
  // merge sorted SA0 suffixes and sorted SA12 suffixes
  for (int p=0,  t=n0-n1,  k=0;  k < n;  k++) {
#define GetI() (SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2)
    int i = GetI(); // pos of current offset 12 suffix
    int j = SA0[p]; // pos of current offset 0  suffix
    if (SA12[t] < n0 ? // different compares for mod 1 and mod 2 suffixes
        leq(T[i],       R[SA12[t] + n0], T[j],       R[j/3]) :
        leq(T[i],T[i+1],R[SA12[t]-n0+1], T[j],T[j+1],R[j/3+n0]))
    {        // suffix from SA12 is smaller
      SA[k] = i;  t++;
      if (t == n02) // done --- only SA0 suffixes left
        for (k++;  p < n0;  p++, k++) SA[k] = SA0[p];
    } else { // suffix from SA0  is smaller
      SA[k] = j;  p++;
      if (p == n0)  // done --- only SA12 suffixes left
        for (k++;  t < n02;  t++, k++) SA[k] = GetI();
    }
  }
  delete [] R; delete [] SA12; delete [] SA0; delete [] R0;
}
```

4.6 Loppuosataulukon sovelluksia

Useimmat loppuosapuun sovellukset voidaan toteuttaa loppuosataulukolla.

Hahmonsovitus

Tarkka hahmonsovitus tehdään tehostetulla binäärihaulla.

- Se on lähes yhtä nopea ($\mathcal{O}(m + \log n + occ)$) kuin hahmonsovitus loppuosapuussa ($\mathcal{O}(m + occ)$).
- Toisin kuin loppuosapuussa, hakuaika pätee, vaikka aakkoston koko ei ole vakio.
- Esiintymien lukumäärä voidaan laskea ajassa $\mathcal{O}(m + \log n)$, sillä binäärihaku tuottaa osavälin, jonka koko on esiintymien lukumäärä. Loppuosapuussa tilanne on hieman monimutkaisempi (HT).

- On olemassa myös aivan toisenlainen hakumenetelmä, takaperinhaku, jolla myös loppuosataulukossa päästään hakuaikaan $\mathcal{O}(m+occ)$ vakioaakkostolla.

Merkkijonon sisäinen rakenne

LCP-arvoista saadaan tietoa jonon rakenteesta.

- Pisimmän toistuvan osajonon pituus on LCP-taulukon maksimi.

- Erilaisten osajonojen lukumäärä saadaan kaavasta.

$$\frac{n(n+1)}{2} + 1 - \sum_{i=0}^{n-1} LCP[i]$$

Yleistetty loppuosataulukko

- Useamman merkkijonon yleistetty loppuosataulukko voidaan määritellä kuten yleistetty loppuosapuu ja sen sovellukset ovat paljolti samat.

- Esimerkiksi kahden jonon S ja T pisin yhteinen osajono löydetään etsimällä jonon $R = S\pounds T\$$ loppuosataulukosta suurin $LCP[i]$, jolla $R_{SA[i]}$ ja $R_{SA[i+1]}$ ovat eri jonoista.

Vahvistettu loppuosataulukko

M. I. Abouelhoda & S. Kurtz & E. Ohlebusch: Replacing Suffix Trees with Enhanced Suffix Arrays. *J. Discrete Algorithms*, 2(1):53–86, 2004.

- Vahvistettu loppuosataulukko lisää normaaliin loppuosataulukkoon ja LCP-taulukkoon vielä pari lisätaulukkoa, jotka tekevät siitä käytännössä loppuosapuun.

- Merkittävin lisä ovat loppuosalinkit, jotka normaalista loppuosataulukosta puuttuvat.

4.6.1 RMQ-esikäsittely

- Taulukon *osaväliminimi* (range minimum) on pienin arvo annetulla taulukon osavälillä. Mikä tahansa taulukko voidaan esikäsitellä lineaarisessa ajassa siten, että jokaiseen osavälikyselyyn voidaan vastata vakioajassa.
- LCP-taulukossa tehtävä osavälikysely (RMQ, range minimum query) antaa kahden loppuosan pisimmän yhteisen alkuosan.

Lemma 4.6.1 *Kahden loppuosan $S_i < S_j$ pisimmän yhteisen alkuosan pituus $lcp(S_i, S_j)$ on* $\min\{LCP[k] \mid SA^{-1}[i] \leq k < SA^{-1}[j]\}$.

Todistus.

- Olkoon $i' = SA^{-1}[i]$ ja $j' = SA^{-1}[j]$.
- Siis $S_i = S_{SA[i']} < S_{SA[i'+1]} < \cdots < S_{SA[j'-1]} < S_{SA[j']} = S_j$.
- Osoitetaan induktiolla $(j' - i')$:n suhteen, että
$$lcp(S_{SA[i']}, S_{SA[j']}) = \min\{LCP[k] \mid i' \leq k < j'\} \ .$$
- Tapaus $j' - i' = 1$ seuraa suoraan LCP:n määritelmästä.
- Kun $j' - i' > 1$, lemman 3.8.2 mukaan
$$lcp(S_{SA[i']}, S_{SA[j']}) = \min\{lcp(S_{SA[i']}, S_{SA[i'+1]}), lcp(S_{SA[i'+1]}, S_{SA[j']})\}$$
ja väite seuraa induktio-oletuksesta.

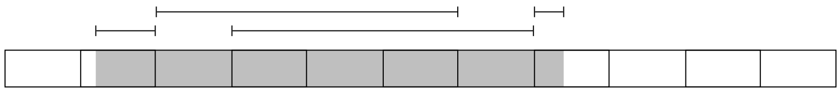
□

- RMQ-esikäsitellyllä loppuosataulukolla voidaan korvata LCA-esikäsitelty loppuosapuu.
- RMQ-esikäsittely on yksinkertaisempi kuin LCA-esikäsittely.

Tarkastellaan, miten RMQ-esikäsittely tehdään taulukolle $L[0, n)$.

- Osavälin $L[i \ldots j]$ *peite* on joukko osavälejä, joiden yhdiste on $L[i \ldots j]$. Osavälin mimini on selvästi sama kuin sen peitteen minimien minimi.
- Osavälimini voidaan siis laskea vakioajassa, jos se voidaan peittää vakiomäärällä osavälejä, joiden minimi tunnetaan.

- Jaetaan taulukko lohkoihin, joiden koko on $\log n$.
- Lasketaan ja talletetaan osaväliminimi jokaiselle osavälille, joka koostuu 2^k lohkosta, jollakin $0 \leq k \leq \log(n/\log n)$.
  - Tällaisia osavälejä on $\mathcal{O}(n)$ kappaletta.
  - Lasketaan yhden lohkon pituisten osavälien minimit taulukon läpikäynnillä. Pidempien osavälien mimini saadaan kahden puolikkaan pituisen osavälin muodostamasta peitteestä. Aikavaatimus on $\mathcal{O}(n)$.
  - Kahdella tällaisella osavälillä voidaan peittää mikä tahansa kokonaisista lohkoista koostuva osaväli.
- Lasketaan ja talletetaan jokaiselle lohkolle sen alkuosien ja loppuosien minimit.
  - Niitä on yhteensä $\mathcal{O}(n)$ kappaletta.
  - Laskemalla minimit nousevassa pituusjärjestyksessä, kunkin alku- tai loppuosan minimi saadaan vakioajassa yhtä lyhyemmän alku- tai loppuosan minimistä.
- Nyt mikä tahansa osaväli, jonka alku ja loppu ovat eri lohkoissa, voidaan peittää neljällä edellä kuvatulla osavälillä, joten sen minimi voidaan laskea vakioajassa.



- Ongelmana on enää kokonaan yhden lohkon sisällä olevat osavälit. Tarkastellaan osaväliä $L[i \dots j]$.
- Olkoon $P(i)$ pienin $k > i$, jolla $L[k] < L[i]$. Tällöin positiosta i alkavan osavälin minimi on jossakin positioista $i, P(i), P(P(i)), P(P(P(i))), \dots$, tarkemmin sanottuna viimeisessä näistä positioista, joka on osavälin sisällä.
- Koodataan ja talletetaan positioon i liittyvä lista $i, P(i), P(P(i)), \dots$ bittivektorina $B(i)$, jossa k:s bitti on ykkönen, jos ja vain jos lohkon alusta lukien k:s positio on listalla.
  - Bittivektorin pituus on $\log n$, joten voidaan olettaa, että se mahtuu $\mathcal{O}(1)$ tietokonesanaan.

- Bittivektori $B(i)$ saadaan vakioajassa vektorista $B(P(i))$.
- Lasketaan arvot $P(i)$ lohkon lopusta alkuun. Voidaan osoittaa, että arvo $P(i)$ on listalla $i+1, P(i+1), P(P(i+1)), \ldots$, ja että lineaarinen haku tältä listalta toimii tasoitetussa vakioajassa.
- Kaikki bittivektorit on siis mahdollista laskea lineaarisessa ajassa.

• Nyt osavälin $L[i \ldots j]$ minimi saadaan seuraavasti:

- Bittivektorista $B(i)$ nollataan bitit, jotka ovat positiosta j oikealle.
- Jäljelle jäävän vektorin oikeanpuoleisin ykkös-bitti kertoo osaväliminimin position.
- Oikeanpuoleisin ykkös-bitti saadaan esimerkiksi katsomalla se $\mathcal{O}(n)$:n kokoisesta taulukosta, johon se on laskettu etukäteen kaikille $\log n$:n pituisille bittivektoreille.

Lause 4.6.2 *Olkoon $L[0 \ldots n)$ taulukko, jonka alkioilla on järjestys. L voidaan esikäsitellä ajassa $\mathcal{O}(n)$ siten, että minkä tahansa osavälin $L[i \ldots j]$, $0 \leq i \leq j < n$, pienin alkio löydetään vakioajassa.*

4.6.2 Burrows–Wheeler-muunnos

M. Burrows & D. Wheeler: A Block Sorting Lossless Data Compression Algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

- Yksi tärkeimmistä tiedontiivistyksen menetelmistä on Burrows-Wheeler-muunnos. Sillä on läheinen yhteys loppuosataulukkoon.

- Tavallisin tapa määritellä merkkijonon S Burrows-Wheeler muunnos on:

  - Olkoon $S[0, n)$ merkkijono. Oletetaan lisäksi, että $S[n] = \$$, missä \$ on kaikkia merkkejä pienempi.
  - S:n rotaatio on jono $S[i, n]S[0, i)$, $0 \leq i \leq n$.
  - Olkoon M $(n+1) \times (n+1)$-matriisi, jonka riveinä ovat S:n kaikki rotaatiot aakkosjärjestyksessä.
  - BW-muunnos $BWT[0, n]$ on M:n viimeinen sarake.

Esimerkki 4.6.3 *Jonon* `banana$` *BW-muunnos on* `annb$aa`*.*

```
M: $banan a     SA: $
   a$bana n         a$
   ana$ba n         ana$
   anana$ b         anana$
   banana $         banana$
   na$ban a         na$
   nana$b a         nana$
```

- Loppumerkin ansiosta rivijärjestys saadaan suoraan loppuosataulukosta. Siten
$$BWT[i] = \begin{cases} \$ & \text{jos } SA[i] = 0 \\ S[SA[i]-1] & \text{muuten} \end{cases}$$

- BW-muunnos on kääntyvä, eli alkuperäinen teksti voidaan palauttaa muunnoksesta.

- Matriisi M saadaan muodostettua seuraavasti.

  - Tarkastellaan matriisia M' joka saadaan M:stä siirtämällä viimeinen sarake ensimmäiseksi (rotaatio oikealle).

– M':n riveinä on edelleen S:n rotaatiot mutta eri järjestyksessä. Siten M' voidaan muuntaa M:ksi paitsi rotaatiolla vasemmalle, myös järjestämällä sen rivit.

– M voidaan nyt muodostaa lähtemällä tyhjästä matriisista ja toistamalla seuraavia askelia, kunnes se on täytetty:

1. Täytä M:n viimeinen sarake BW-muunnoksesta.
2. Muunna M M':ksi rotaatiolla oikealle.
3. Muunna M' M:ksi järjestämällä rivit.

```
M: ??????a     M’: a??????     M: $??????
   ??????n         n??????        a??????
   ??????n         n??????        a??????
   ??????b         b??????        a??????
   ??????$         $??????        b??????
   ??????a         a??????        n??????
   ??????a         a??????        n??????

M: $?????a     M’: a$?????     M: $b?????
   a?????n         na?????        a$?????
   a?????n         na?????        an?????
   a?????b         ba?????        an?????
   b?????$         $b?????        ba?????
   n?????a         an?????        na?????
   n?????a         an?????        na?????

M: $b????a     M’: a$b????     M: $ba????
   a$????n         na$????        a$b????
   an????n         nan????        ana????
   an????b         ban????        ana????
   ba????$         $ba????        ban????
   na????a         ana????        na$????
   na????a         ana????        nan????
```

- Palautus voidaan tehdä myös ilman matriisia M tilassa $\mathcal{O}(n)$.

  – Järjestetään BWT:n merkit vakaasti. Olkoon f näin saatu permutaatio eli $BWT[f(i)] \leq BWT[f(i+1)]$ ja yhtäsuuruuden tapauksessa $f(i) < f(i+1)$.

```
        0123456              i: 0123456
   BWT: annb$aa           f(i): 4056312
                     BWT[f(i)]: $aaabnn
```

– f on myös permutaatio M:n M':n rivien välillä.

– Alkuperäinen teksti saadaan säännöllä

$$S[i] = BWT[f^{(i+2)}(0)] ,$$

missä $f^{(k)}$ tarkoittaa f:n toistoa k kertaa: $f(f(f(\cdots f(k)\cdots)))$.

– Myös loppuosataulukko saadaan helposti BWT:stä:

$$SA^{-1}[i] = f^{(i+1)}(0) ,$$

Esimerkki 4.6.4 *Kun* $BWT =$ *annb$aa, niin*

$$\begin{aligned} &f(0) = 4,\ f(4) = 3,\ f(3) = 6,\ f(6) = 2, \\ &f(2) = 5,\ f(5) = 1,\ f(1) = 0,\ f(0) = 4. \end{aligned}$$

Siten

$$\begin{aligned} S &= BWT[3]BWT[6]BWT[2]BWT[5]BWT[1]BWT[0]BWT[4] \\ SA^{-1} &= (4, 3, 6, 2, 5, 1, 0) \end{aligned}$$

BW-muunnosta käytetään tiedontiivistyksessä.

- Muunnettu teksti on helpommin tiivistyvä kuin alkuperäinen, koska se sisältää paljon paikallista toistoa.

  – Esim. englanninkielisessä tekstissä `he`-alkuiset rotaatiot tulevat järjestämisen jälkeen peräkkäin, jolloin viimeisessä sarakkeessa on vastaavalla kohdalla paljon kirjaimia `t` ja `T`.

- BW-muunnokseen perustuvat myös useat tiivistetyt loppuosarakenteet.