proseduurikutsulla ja kutsun aikana proseduurin silmukka suoritettiin $k > 0$ kertaa, d' ei pienene vaan kasvaa $k - 1$:llä. Siten proseduurin silmukka suoritetaan enintään $\mathcal{O}(n)$ kertaa.

□

4.3 Loppuosapuun sovelluksia

Esitetään seuraavaksi joitakin esimerkkejä loppuosapuiden monista sovelluksista.

Hahmonsovitus

Aiemmin todettiin jo, että loppuosapuu mahdollistaa nopean hahmonsovituksen.

- Vaikka itse loppuosapuun muodostus laskettaisiin, on tarkka hahmonsovitus asymptoottisesti yhtä nopeaa kuin Knuth–Morris–Pratt-algoritmilla yhden hahmon tapauksessa tai Aho–Corasick-algoritmilla monen hahmon tapauksessa.
- Varsinainen hyöty saadaan, jos loppuosapuu voidaan muodostaa etukäteen.
- Loppuosapuu soveltuu myös likimääräiseen hakuun käyttäen seulontamenetelmiä, esimerkiksi luvussa 2.8 kuvattua menetelmää, joka perustuu hahmon jakoon $k + 1$ palaseen ja palasten tarkkaan hahmonsovitukseen.

Merkkijonon sisäinen rakenne

Loppuosapuu paljastaa myös paljon tietoa jonon S sisäisestä rakenteesta.

- Jonon S *erilaisten osajonojen lukumäärä* on täsmälleen loppuosatrien (siis loppuosien joukon tiivistämättömän trien) solmujen lukumäärä. Loppuosapuusta tämä luku saadaan laskemalla yhteen kaarten nimien pituudet ja lisäämällä yksi (juuri/tyhjä jono). Aikavaatimus on $\mathcal{O}(n)$, vaikka tulos on yleisesti $\Theta(n^2)$.
- Merkkijonon S *pisin toistuva osajono* on pisin jono, joka esiintyy vähintään kahdesti S:ssä. Loppuosapuun syvin sisäsolmu esittää pisintä toistuvaa osajonoa.

Yleistetty loppuosapuu

Loppuosapuu soveltuu myös kahden tai useamman jonon vertailemiseen.

- Kahden merkkijonon S ja T *yleistetty loppuosapuu* on jonon $S\pounds T\$$ loppuosapuu. Huomaa, että S:n alueelta alkavien loppuosien yksilöivä alkuosa päättyy erotinmerkkiin £. Usemman jonon tapauksessa kaikkien erotinmerkkien täytyy olla erilaisia. Yksittäisen merkin sijasta erottimina voi olla myös jonoja, jotka takaavat, että loppuosien yksilöivät alkuosat eivät ulotu erottimien yli.

- Merkitään jokaiseen lehteen, alkaako vastaava loppuosa S:stä vai T:stä. Syvyyssuuntaisella läpikäynnillä voidaan jokaiselle sisäsolmulle laskea, sisältääkö sen alipuu S-lehtiä, T-lehtiä vai molempia.

- Nyt syvin sisäsolmu, jonka alipuu sisältää molempia, esittää S:n ja T:n *pisintä yhteistä osajonoa*.

- Siis pisin yhteinen osajono voidaan laskea lineaarisessa ajassa.

Loppuosapuu Aho-Corasick-automaattina

- Jonon S loppuosalinkeillä varustettu loppuosatrie on Aho-Corasick-automaatti, kun loppuosalinkit toimivat korjaussiirtyminä.

- Myös loppuosapuuta voidaan käyttää AC-automaattina. Korjaussiirtymä keskeltä kaarta tehdään kuten proseduurissa Set-slink() (mutta luomatta uusia solmuja). Tämän operaation tasattu aikavaatimus on $\mathcal{O}(1)$ samalla argumentilla kuin loppuosapuun muodostusalgoritmissa.

- Muodostetaan jonon S loppuosapuu ja selataan sillä jonoa T kuten AC-automaatilla.

- Jos ennen merkin $T[j]$ lukemista ollaan syvyydellä d, se tarkoittaa, että jollakin i, $S[i \dots i+d) = T[j-d \dots j)$. Jos $T[j]$ aiheuttaa korjaussiirtymän, $T[j-d \dots j)$ on pisin T:n positiosta $j-d$ alkava osajono, joka esiintyy S:ssä.

- Korjaussiirtymä kasvattaa $j-d$:tä yhdellä ja siirtymä puussa alaspäin ei muuta $j-d$:tä. Siis selauksen aikana saadaan kaikkien T:n loppuosien pisimmät alkuosat, jotka esiintyvät S:ssä.

- Niistä pisin on S:n ja T:n pisin yhteinen osajono. Saatiin siis vaihtoehtoinen algoritmi pisimmän yhteisen osajonon laskemiseen.

LCA-esikäsittely

Loppuosapuun lisäkäsittelyllä saadaan vielä lisää sovelluksia.

- Kahden solmun u ja v *lähin (alin/syvin) yhteinen esi-isä* (LCA, lowest common ancestor, tai NCA, nearest common ancestor) on syvin solmu, joka on sekä u:n että v:n esi-isä. Mikä tahansa puu on mahdollista esikäsitellä siten, että kahden solmun lähin yhteinen esi-isä löydetään vakioajassa. Ohitetaan tässä yksityiskohdat, mutta myöhemmin esitetään vastaava tulos loppuosataulukolle tarkemmin.

- Loppuosapuussa kahden lehden lähin yhteinen esi-isä esittää lehtien edustamien loppuosien *pisintä yhteistä alkuosaa*. Se voidaan siis laskea vakioajassa mille tahansa loppuosille, kun loppuosapuu on LCA-esikäsitelty.

- LCA-esikäsittelyä voidaan käyttä esim. palindromien etsimiseen.
  - Palindromi on jono, joka on oma käänteisjononsa. Esimerkiksi `SAIPPUAKAUPPIAS` on palindromi.
  - Kuvataan, miten lineaarisessa ajassa löydetään pisin parittoman mittainen palindromi, joka esiintyy jonossa S. Pisin parillisen mittainen palindromi löydetään vastaavasti.
  - Olkoon S^R jonon S käänteisjono. Jos $S[i-k\ldots i+k]$ on pisin palindromi, jonka keskikohta on i, niin loppuosien $S[i\ldots n)$ ja $S^R[n-i-1\ldots n)$ pisin yhteinen alkuosa on $S[i\ldots i+k] = S^R[n-i-1\ldots n-i-1+k]$.
  - Se löydetään vakioajassa jonojen S ja S^R yleistetystä loppuosapuusta, joka on LCA-esikäsitelty. Vastaava kysely voidaan tehdä kaikille mahdollisille keskikohdille ajassa $\mathcal{O}(n)$.

$O(kn)$-algoritmi likimääräiseen hahmonsovitukseen

- Kertaa lävistäjämenetelmä (Luku 2.3).

- Periaatteessa lävistäjämenetelmän yleistäminen likimääräiseen hahmonsovitukseen onnistuu simuloimalla ensimmäisen rivin asettamista 0:ksi:
  - Nyt ei kuitenkaan lävistäjien vyötä voida rajoittaa tarpeeksi, koska esiintymät voivat alkaa miltä sarakkeelta tahansa.

  - Algoritmin aikavaatimukseksi näin toteutettuna tulee $O(mn)$, eli sama kuin perusratkaisussa.

- Huomataan kuitenkin, että saadaan nopeampi algoritmi, jos alirutiini Algoritmi A voidaan suorittaa vakioajassa.
  - Likimääräinen hahmonsovitus ajassa $O(kn)$.
  - Etäisyyden $d = D(A, B)$ laskeminen ajassa $\mathcal{O}(d^2 + n)$.

- Algoritmin A toimii vakioajassa lukuunottamatta riviä

  (2) **while** $t < m$ **and** $t + k < n$ **and** $a_{t+1} = b_{t+1+k}$ **do** $t := t + 1$;

- Merkataan

  $$JUMP(i, j) = \min\{\ell \geq 0 \mid i + \ell = \min(m, n - k) + 1 \text{ tai } a_{i+\ell} \neq b_{j+\ell}\},$$

  ja huomataan, että rivi (2) voidaan korvata rivillä

  (2') $t := t + JUMP(t + 1, t + 1 + k)$;

- Jos $JUMP(i, j)$ osataan laskea vakioajassa kaikille i, j, niin Algoritmi A toimii vakioajassa.

- Tutkitaan nyt miten $JUMP(i, j)$ voidaan toteuttaa vakioajassa.

Yleistetty loppuosapuu

- Huomataan, että $JUMP(i, j)$ laskee loppuosien A_i ja B_j pisimmäm yhteisen alkuosan pituuden $lcp(A_i, B_j)$.

- Se voidaan siis laskea vakioajassa LCA-esikäsitellystä jonojen A ja B yleistetystä loppuosapuusta.

- Käytännössä yleistetyn loppuosapuun muodostus ja sen esikäsittely on varsin raskas operaatio, erityisesti likimääräisessä hahmonsovituksessa, kun teksti on pitkä. Esitetään seuraavaksi kevyempi menetelmä.

Vain hahmon loppuosapuu

- Oletetaan nyt, että $A = P$ ja $B = S$.
- Muodostetaan hahmon P loppuosapuu ja tehdään sen LCA-esikäsittely.
- Selataan teksti S käyttäen loppuosapuuta AC-automaattina. Saadaan jokaiselle loppuosalle S_j sen pisin alkuosa, joka esiintyy hahmossa; Olkoon $I(j)$ esiintymän alkukohta.
- Nyt $JUMP(i, j) = lcp(P_i, S_j) = \min\{lcp(P_i, P_{I(j)}), lcp(P_{I(j)}, S_j)\}$, mikä voidaan laskea vakioajassa, kun arvot$I(j)$ on talletettu.

Lause 4.3.1 *Merkkijonojen $A = a_1 \cdots a_m$ ja $B = b_1 \cdots b_n$, $m \leq n$, editointietäisyys $d = D(A, B)$ voidaan laskea tehostetulla lävistäjämenetelmällä ajassa $O(d^2 + n)$.*

Lause 4.3.2 *Olkoon hahmo $P = p_1 \cdots p_m$ ja teksti $S = s_1 \cdots s_n$, $m \ll n$. Tehostetulla lävistäjämenetelmällä likimääräinen hahmonsovitus k virheellä voidaan ratkaista ajassa $O(kn)$.*

4.4 Loppuosataulukko

U. Manber & G. Myers: Suffix Arrays: A New Method for On-line String Searches. *SIAM Journal on Computing* 22(5):935–948, 1993.

- Merkkijonon S loppuosataulukko on loppuosien joukon $S_{[0,n]}$ järjestetty taulukko.
- Täsmällisemmin, S:n loppuosataulukko on taulukko $SA[0 \ldots n]$, joka sisältää joukon $[0, n]$ permutaation ja toteuttaa ehdon $S_{SA[0]} < S_{SA[1]} < \cdots < S_{SA[n]}$.
- Hahmonsovitus eli alkuosahaku hahmolla P voidaan tehdä ajassa $\mathcal{O}(|P| + \log n + occ)$ binäärihaulla olettaen, että myös tehostetussa binäärihaussa tarvittavat LCP_l ja LCP_r taulukot on annettu.
- Monissa loppuosataulukon sovelluksissa on keskeisessä asemassa toinen LCP-taulukko: $LCP[i] = lcp(S_{SA[i]}, S_{SA[i+1]})$.

- Samoin usein esiintyvä taulukko on loppuosataulukon käänteistaulukko: $SA^{-1}[i] = j$ jos ja vain jos $SA[j] = i$.
- $SA^{-1}[i]$ voidaan myös tulkita loppuosan S_i järjestysnumeroksi.

Esimerkki 4.4.1 *Jonon S = `banana` loppuosataulukko, LCP-taulukko ja loppuosataulukon käänteistaulukko.*

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

	SA^{-1}	
0	4	`banana`
1	3	`anana`
2	6	`nana`
3	2	`ana`
4	5	`na`
5	1	`a`
6	0	

LCP-taulukon laskeminen

T. Kasai, G. Lee, H. Arimura, S. Arikawa & K. Park: Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. Teoksessa *Proc. CPM 2001*, LNCS 2089, sivut 181–192, 2001.

- Käänteistaulukko SA^{-1} on helppo laskea lineaarisessa ajassa loppuosataulukosta SA.
- Sen avulla myös LCP-taulukko voidaan helposti laskea lineaarisessa ajassa

Lemma 4.4.2 *Kaikilla $0 \leq i < n$, $LCP[SA^{-1}[i+1]] \geq LCP[SA^{-1}[i]] - 1$.*

Todistus.

- Oletetaan, että S_i ja S_j ovat aakkosjärjestyksessä peräkkäin eli $LCP[SA^{-1}[i]] = lcp(S_i, S_j) = \ell$.
- Jos $\ell > 0$, niin jollakin merkillä a, $S_i = aS_{i+1}$ ja $S_j = aS_{j+1}$ ja siten $S_{i+1} < S_{j+1}$ ja $lcp(S_{i+1}, S_{j+1}) = \ell - 1$.
- Tällöin $LCP[SA^{-1}[i+1]] \geq lcp(S_{i+1}, S_{j+1}) = \ell - 1$.

□

Seuraava algoritmi laskee LCP-taulukon järjestyksessä $LCP[SA^{-1}[0]]$, $LCP[SA^{-1}[1]]$, $LCP[SA^{-1}[2]]$, ...

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) $\ell := 0$;
(2) **for** $i := 0$ **to** $n-1$ **do**
(3) $\quad k := SA^{-1}[i]$; $\qquad$ (* $i = SA[k]$ *)
(4) $\quad$ **if** $k = n$ **then** $LCP[k] := 0$;
(5) $\quad$ **else**
(6) $\qquad j := SA[k+1]$;
(7) $\qquad \ell := \ell + lcp(S_{i+\ell}, S_{j+\ell})$;
(8) $\qquad LCP[k] := \ell$; $\qquad$ (* $LCP[k] := lcp(S_i, S_j)$ *)
(9) $\quad$ **if** $\ell > 0$ **then** $\ell := \ell - 1$;

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.

□

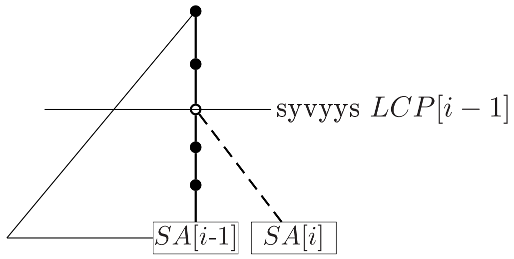
Loppuosapuun ja -taulukon suhde

- Loppuosataulukko voidaan muodostaa loppuosapuusta yksinkertaisesti käymällä lehdet läpi aakkosjärjestyksessä.
- Myös LCP-taulukko voidaan laskea helposti syvyyssuuntaisen läpikäynnin aikana:

  - Kahden lehden lähin yhteinen esi-isä on ylin solmu, jossa vieraillaan lehtien välissä.
  - Lähimmän yhteisen esi-isän syvyys antaa pisimmän yhteisen alkuosan pituuden eli LCP-arvon.

Myös muunnos loppuosa- ja LCP-taulukosta loppuosapuuksi on yksinkertainen.

- Lisätään loppuosat puuhun aakkosjärjestyksessä.
- Kuten triviaalialgoritmissa McCreightin algoritmissa, kutakin loppuosaa kohden luodaan lehti ja kiinnitetään se puuhun. Kiinnityskohtaan voidaan joutua lisäämään uusi solmu.
- Tarkastellaan tilannetta, jossa $S_{SA[i-1]}$ on juuri lisätty ja seuraavaksi lisätään $S_{SA[i]}$.
- Uudesta lehdestä u tulee puun oikeanpuoleisin lehti. Siis kiinnityskohdan täytyy olla puun oikeanpuolimmaisella polulla, joka johtaa loppuosaa $S_{SA[i-1]}$ edustavaan lehteen u'.
- Kiinnityskohdassa olevasta solmusta v tulee u':n ja u:n lähin yhteinen esi-isä. Siis sen syvyys on $LCP[i-1]$.



- Pidetään oikeanpuoleisen polun solmut pinossa lehti huipulla. Uuden lehden kiinnitys tehdään seuraavasti:
  - Poistetaan pinon huipulta solmut, jotka ovat kiinnityskohtaa syvemmällä. Tällöin pinon huipulle jäävä solmu on kiinnityskohta tai kiinnityskohtaan tulevan uuden solmun vanhempi.
  - Kiinnityksen jälkeen mahdollinen uusi sisäsolmu ja uusi lehti lisätään pinoon.
- Kukin solmu lisätään pinoon kerran. Jokainen kiinnityskohtaa etsittäessä ohitettu solmu poistetaan pinosta eikä sitä enää lisätä takaisin.
- Aikavaatimus on siis $\mathcal{O}(n)$.

4.5 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).
- Myöhemmin esitetään toinen hakumenetelmä, jolla myös loppuosataulukossa päästään hakuaikaan $\mathcal{O}(m + occ)$ vakioaakkostolla.

Merkkijonon sisäinen rakenne

LCP-arvoista saadaan tietoa jonon rakenteesta.

- Edellä kuvattu loppuosapuun muodostaminen loppuosa- ja LCP-taulukosta paljastaa, miten loppuosapuun kaarten pituuksien summa eli erilaisten osajonojen määrä voidaan laskea. Kun loppuosa $S_{SA[i]}$ lisätään, kaarten yhteispituus kasvaa uuteen lehteen johtavan kaaren pituudella $n - SA[i] - LCP[i - 1]$.
- Pisimmän toistuvan osajonon pituus on LCP-taulukon maksimi.

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 $R = S£T\$$ loppuosataulukosta suurin $LCP[i]$, jolla $R_{SA[i]}$ ja $R_{SA[i+1]}$ ovat eri jonoista.

Loppuosalinkit puuttuvat

- Loppuosalinkit puuttuvat loppuosataulukosta eikä niitä saa siihen luontevasti lisättyä. (Epäluontevia ratkaisuja toki on.)
- Siten loppuosataulukkoa ei esimerkiksi voi käyttää AC-automaattina.
- Monille loppuosalinkkejäkin käyttäville sovelluksille on kuitenkin olemassa vaihtoehtoisia menetelmiä, jotka toimivat loppuosataulukossa.

4.5.1 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 4.5.1 *Kahden loppuosan $S_i < S_j$ pisimmän yhteisen alkuosan pituus $lcp(S_i, S_j)$ on* $\min\{LCP[k] \mid SA^{-1}[i] \le k < SA^{-1}[j]\}$.

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

$$lcp(S_{SA[i']}, S_{SA[j']}) = \min\{LCP[k] \mid i' \le k < j'\} .$$

- Tapaus $j' - i' = 1$ seuraa suoraan LCP:n määritelmästä.
- Kun $j' - i' > 1$, lemman 3.8.2 mukaan

$$lcp(S_{SA[i']}, S_{SA[j']}) = \min\{lcp(S_{SA[i']}, S_{SA[i'+1]}), lcp(S_{SA[i'+1]}, S_{SA[j']})\}$$

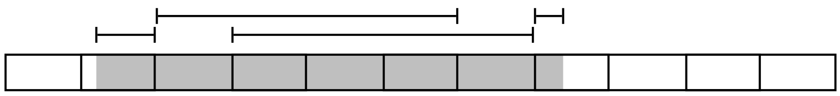
ja väite seuraa induktio-oletuksesta.

□

- 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 $L[i \dots j]$ *peite* on joukko osavälejä, joiden yhdiste on $L[i \dots j]$. 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 $i, P(i), P(P(i)), P(P(P(i))), \ldots$, tarkemmin sanottuna viimeisessä näistä positioista, joka on osavälin sisällä.

- Koodataan ja talletetaan positioon i liittyvä lista $i, P(i), P(P(i)), \ldots$ 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 4.5.2 *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.*

4.5.2 Takaperinhaku

P. Ferragina & G. Manzini: Opportunistic Data Structures with Applications. Teoksessa *Proc. IEEE Symposium on Foundations of Computer Science*, sivut 390–398, 2000.

- Esitetään loppuosataulukolle $\mathcal{O}(m + occ)$ ajassa toimiva hakualgoritmi.

- Haku loppuosapuussa ja yleensä triessä perustuu siihen, että jos on annettu jonoa X esittävä kohta puussa ja merkki a, niin jonoa Xa esittävä kohta löydetään vakioajassa.

- Takaperinhaku (backward search) loppuosataulukossa tekee saman takaperin: Annettuna hakutulos jonolle X ja merkki a, hakutulos jonolle aX saadaan vakioajassa.

- Hakutulos on positio tai osaväli loppuosataulukossa. Oletetaan jatkossa, että hakutulos on pienin i, jolla $P \leq S_{SA[i]}$. (Jos $P > S_{SA[n]}$, tulos on $n+1$ eli $S_{SA[n+1]}$ tulkitaan kaikkia jonoja suuremmaksi.) Siis, jos P esiintyy S:ssä, $S_{SA[i]}$ on aakkosjärjestyksessä pienin loppuosa, jonka alkuosa on P.

- Olkoon $B(i,a)$ hakutulos jonolla $aS_{SA[i]}$. Talletetaan $B(i,a)$ kaikilla $0 \leq i \leq n+1$ ja $a \in \Sigma$.

- Saadaan seuraava algoritmi.

Algoritmi 4.5.3 *Takaperinhaku*

Syöte: Merkkijono $S[0 \ldots n)$, taulukko B ja hakujono $P[0 \ldots m)$.
Tuloste: i, jolla $S_{SA[i-1]} < P \leq S_{SA[i]}$ tai
0, jos $P \leq S_{SA[0]}$; tai $n+1$, jos $S_{SA[n]} < P$;

$i := 0$;
for $k := n-1$ **downto** 0 **do**
    $i := B(i, P[k])$;   (* i on hakutulos P_k:lle *)
return i;

Lemma 4.5.4 *Algoritmi 4.5.3 toimii oikein.*

Todistus. Osoitetaan, että kussakin askeleessa laskettava i on oikea hakutulos jonolle P_k.

- Tulos $i = 0$ on oikein jonolle $P_m = \varepsilon$ (tyhjä jono).

- Tehdään vastaoletus, että haku menee vikaan ensimmäisen kerran P_k:lle.

- Olkoon i' (oikea) hakutulos P_{k+1}:lle ja i (väärä) hakutulos P_k:lle.

- Tällöin siis $S_{SA[i'-1]} < P_{k+1} \leq S_{SA[i']}$. (Ohitetaan tässä tapaukset $i' = 0$ ja $i' = n+1$.)

- Toisaalta B:n määritelmän mukaan $S_{SA[i-1]} < aS_{SA[i']} \leq S_{SA[i]}$, missä $a = P[k]$.
- Jos i on väärä tulos P_k:lle, pätee joko
  1. $P_k \leq S_{SA[i-1]} < aS_{SA[i']}$ tai
  2. $aS_{SA[i']} \leq S_{SA[i]} < P_k$.
- Koska P_k ja $aS_{SA[i']}$ molemmat alkavat a:lla, myös niiden välissä oleva jonon täytyy alkaa a:lla. Tällöin a:n poistaminen alusta ei muuta järjestystä. Siis joko
  1. $P_{k+1} \leq S_{SA[i-1]+1} < S_{SA[i']}$ tai
  2. $S_{SA[i']} \leq S_{SA[i]+1} < P_{k+1}$.
- Molemmat ovat vastoin oletusta, että P_{k+1}:n hakutulos oli oikein.

□

- Taulukon B tilavaatimus on $\mathcal{O}(n\sigma)$ eli vakioaakkostolla $\mathcal{O}(n)$.
- $B(*, a)$ voidaan laskea ajassa $\mathcal{O}(n)$ seuraavasti:
  - Olkoon $SA[l \ldots r]$ a:lla alkavien loppuosien muodostama osaväli.
  - Jos $l < i \leq r$, niin $aS_{SA[i-1]+1} = S_{SA[i-1]} < S_{SA[i]} = aS_{SA[i]+1}$ ja siten $S_{SA[i-1]+1} < S_{SA[i]+1}$
  - Siis $SA[i-1]+1$ esiintyy loppuosataulukossa ennen arvoa $SA[i]+1$. Jos arvo j on niiden välissä, pätee
  $$S_{SA[i-1]+1} < S_j < S_{SA[i]+1} \ ,$$
  joten täytyy olla
  $$S_{SA[i-1]} < aS_j < S_{SA[i]} \ .$$
  Tällöin $B(j, a) = i$.
  - Lasketaan $B(*, a)$ selaamalla loppuosataulukkoa ja etsimällä arvoja $SA[i] + 1$, $l \leq i \leq r$. Niiden väliin jäävät arvot sijoitetaan yllä kuvatulla tavalla.

Lause 4.5.5 *Jono $S[0 \ldots n)$ vakioaakkostossa ja sen loppuosataulukko voidaan esikäsitellä ajassa $\mathcal{O}(n)$ niin, että takaperinhaku jonolla P, $|P| = m$, voidaan toteuttaa ajassa $\mathcal{O}(m)$.*

- Taulukon sijasta $B(i, a)$ voidaan toteuttaa myös muunlaisella tietorakenteella. Esimerkiksi suurellakin aakkostolla se voidaan toteutta tilassa $\mathcal{O}(n)$ siten, että se voidaan laskea ajassa $\mathcal{O}(\log \sigma)$.

- Takaperinhaku on erityisen tärkeä tekniikka tiivistetyissä tekstin hakurakenteissa, kuten tiivistetyssä loppuosataulukossa. Niissä käytetään erittäin monimutkaisia $B(i, a)$:n toteutuksia.

4.5.3 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 4.5.6 *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

$$BWT[i] = \begin{cases} \texttt{\$} & \text{jos } SA[i] = 0 \\ S[SA[i] - 1] & \text{muuten} \end{cases}$$

- 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: $f(f(f(\cdots f(k)\cdots)))$.

Esimerkki 4.5.7 *Kun $BWT =$ `annb$aa`, niin*

$$f(0) = 4,\ f(4) = 3,\ f(3) = 6,\ f(6) = 2,$$
$$f(2) = 5,\ f(5) = 1,\ f(1) = 0,\ f(0) = 4.$$

Siten

$$S = BWT[3]BWT[6]BWT[2]BWT[5]BWT[1]BWT[0]BWT[4].$$

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.

- Tällä kurssilla tiedontiivistystä ei käsitellä tarkemmin, mutta keväällä 2006 on tulossa erikoiskurssi tiedontiivistämisestä.