2.4 Yleinen editointietäisyys

Otetaan käyttöön yleinen editointioperaatiojoukko G ja operaatioille yleiset kustannukset.

Määritelmä 2.4.1 *Olkoon $x, y \in \Sigma^*$, $x \neq y$. Silloin $x \to y$ on editointioperaatio. Operaatio korvaa editoitavassa jonossa osajonon x jonolla y. Eli jos $A = axb$, saadaan A:sta editointioperaatiota $x \to y$ soveltamalla ayb. Operaation kustannusta merkitään $w(x \to y)$. Kustannus $w(x \to y)$ voi siis riippua x:stä ja y:stä. Oletamme, että aina $w(x \to y) \geq 0$.*

Esimerkki 2.4.2 *Operaatiot M, L, ja P voidaan esittää operaatiojoukkona $G_0 = \{x \to y \mid x, y \in \Sigma \cup \{\epsilon\}, x \neq y\}$, jolla on voimassa $w(x \to y) = 1$.*

Editointietäisyys yleistyy ilmeisellä tavalla: Jonojen A ja B välinen editointietäisyys on pienin mahdollinen sellainen editointioperaatiojonon kustannus, että tämä jono muuntaa A:n B:ksi. Täsmennetään tämä seuraavasti:

- Olkoon G yleinen editointioperaatiojoukko ja $A = a_1 \cdots a_m$, $B = b_1 \cdots b_n \in \Sigma^*$. Jaetaan A ja B osiin $A = x_1 \cdots x_h$, $B = y_1 \cdots y_h$ siten, että kukin $x_i, y_i \in \Sigma^*$ ja kukin $x_i \to y_i \in G$ tai $x_i = y_i$.

Määritelmä 2.4.3 *$T = (x_1 \to y_1, \ldots, x_h \to y_h)$ on editointiketju G:ssä jonosta A jonoon B. Editointiketjun T kustannus on $w(T) = \sum_{i=1}^{h} w(x_i \to y_i)$.*

Määritelmä 2.4.4 *Jonojen A ja B välinen yleinen editointietäisyys G:n suhteen on $D_G(A, B) = \min\{w(T) \mid T \text{ on editointiketju } A\text{:sta } B\text{:hen}\}$.*

Huomaa, että editointietäisyyttä laskettaessa kutakin A:n osaa saadaan muuttaa editointioperaatioilla korkeintaan kerran.

- Jos operaatioita $x \to y$ sallitaan käytettävän kuten kontekstittomien kielioppien produktioita, silloin on mahdollista, että sama A:n osa muuttuu useita kertoja.

Määritelmä 2.4.5 *$T' = (x_1 \to y_1, \ldots, x_h \to y_h)$ on vapaa editointiketju G:ssä jonosta A jonoon B, jos on olemassa merkkijonot α_i, β_i, $i = 1, \ldots, h$, siten, että*

$$\begin{aligned} A = A_1 &= \alpha_1 x_1 \beta_1 \\ A_2 &= \alpha_1 y_1 \beta_1 = \alpha_2 x_2 \beta_2 \\ A_3 &= \alpha_2 y_2 \beta_2 = \alpha_3 x_3 \beta_3 \\ &\cdots \\ A_{h-1} &= \alpha_{h-1} y_{h-1} \beta_{h-1} = \alpha_h x_h \beta_h \\ A_h &= \alpha_h y_h \beta_h = B. \end{aligned}$$

Ketjun T' kustannus on $w(T') = \sum_{i=1}^{h} w(x_i \to y_i)$.

Määritelmä 2.4.6 *Jonojen A ja B välinen vapaa editointietäisyys G:n suhteen on $D'_G(A, B) = \min\{w(T') \mid T'$ on vapaa editointiketju A:sta B:hen$\}$.*

Huomautus. Pätee $D'_G(A, B) \leq D_G(A, B)$. Erisuuruus voi olla aito.

Esimerkki 2.4.7

$$\begin{array}{llllllll} G: \{ & \mathtt{a} \to \mathtt{b}, & \mathtt{b} \to \mathtt{c}, & \mathtt{a} \to \mathtt{c}, & \mathtt{a} \to \mathtt{e}, & \mathtt{e} \to \mathtt{a}, & \mathtt{e} \to \mathtt{c} & \} \\ w: & 1 & 1 & 3 & 2 & 2 & 2 & \end{array}, \quad \textit{jolloin}$$

$D'_G(\mathtt{a}, \mathtt{c}) = 2 = w(\mathtt{a} \to \mathtt{b}) + w(\mathtt{b} \to \mathtt{c}) < w(\mathtt{a} \to \mathtt{c}) = 3 = D_G(\mathtt{a}, \mathtt{c})$.

Esimerkin 2.4.2 joukolle G_0 on helppo antaa ehto, joka takaa, että D_{G_0} ja D'_{G_0} samat:

Lause 2.4.8 *Olkoon $G_0 = \{x \to y \mid x, y \in \Sigma \cup \{\epsilon\}, x \neq y\}$. Jos w toteuttaa kolmioepäyhtälön: $w(x \to z) \leq w(x \to y) + w(y \to z)$ kaikilla $x \to z$, $x \to y$, $y \to z \in G_0$, silloin $D'_{G_0}(A, B) = D_{G_0}(A, B)$ kaikilla $A, B \in \Sigma^*$.*

Edellisen esimerkin w ei toteuta kolmioepäyhtälöä. Jos asetetaan $w(\mathtt{a} \to \mathtt{c}) = 2$, kolmioepäyhtälö tulee voimaan.

Huomautus 2.4.9 *Vapaan editointietäisyyden $D'_G(A, B)$ laskeminen on yleisessä tapauksessa algoritmisesti ratkeamaton ongelma! Tämä johtuu siitä, että jopa ongelma "Voidaanko A muuttaa B:ksi G:n operaatioilla" on algoritmisesti ratkeamaton. Mutta jos G ja w ovat tarpeeksi siistejä, algoritmin laatiminen voi onnistua.*

Yleinen editointietäisyys $D_G(A, B)$ voidaan sen sijaan aina laskea dynaamisella ohjelmoinnilla:

- Olkoon jälleen $d_{ij} = D_G(a_1 \cdots a_i, b_1 \cdots b_j)$. Silloin (d_{ij}), $0 \le i \le m$, $0 \le j \le n$, voidaan laskea palautuskaavasta:

$$\begin{aligned} d_{ij} = \quad \min(\quad & (\textbf{if } a_i = b_j \textbf{ then } d_{i-1,j-1} \textbf{ else } \infty), \min\{d_{i-r,j-s} + w(x \to y) \\ & \mid x \to y \in G, A_{1\ldots i-r}x = A_{1\ldots i}, B_{1\ldots j-s}y = B_{1\ldots j}\}). \end{aligned} \tag{2.5}$$

- Alkuehto: $d_{00} = 0$, $d_{ij} = \infty$, $i, j < 0$.

Algoritmi 2.4.10 *$D_G(A, B)$ dynaamisella ohjelmoinnilla.*

Syöte: $A = a_1 \cdots a_m$, $B = b_1 \cdots b_n$, operaatiot G, kustannukset w.
Tuloste: (d_{ij}); erityisesti $d_{mn} = D_G(A, B)$.

```
(1)  d00 := 0;
(2)  for j := 0 to n do
(3)      for i := 0 to m do begin (* if (i,j) ≠ (0,0) *)
(4)          C := ∞;
(5)          for each x → y ∈ G do
(6)              if a_{i-r+1} ··· a_i = x and b_{j-s+1} ··· b_j = y then
(7)                  C := min(C, d_{i-r,j-s} + w(x → y));
(8)          if a_i = b_j then C := min(C, d_{i-1,j-1});
(9)          d_ij := C end end;
```

- Jos riveillä 7 tai 8 viitataan sellaiseen d_{ij}, jota ei ole laskettu edellä, oletetaan, että $d_{ij} = \infty$.

- Algoritmin 2.4.10 analyysi:

  - Algoritmi vaatii tilaa $O(mn)$, missä mn on (d_{ij}):n koko.

  - Algoritmin aikavaatimus on $O(||G||mn)$, sillä se käsittelee mn alkioita d_{ij}, joista kunkin laskenta vie ajan $O(||G||)$ (G:n läpikäynti, rivit 5–7), missä $||G|| = \sum_{x \to y \in G} |x, y|$ on G:n operaatioiden yhteispituus.

- Sisin silmukka (rivit 5-7) voidaan ohjelmoida tehokkaamminkin, jos G on sopiva. Näin tehtiin G_0:n tapauksessa (Alg. 2.2.3).

2.4.1 Lyhimmän kiertotien menettely

Lävistäjälemma ei yleisesti päde $D_G(A, B)$:lle. Matriisin (d_{ij}) laskenta-alaa voidaan kuitenkin myös yleisen editointietäisyyden tapauksessa rajoittaa tehokkaasti.

- Olkoon *minimoiva polku* matriisin (d_{ij}) ne solut, joihin liittyy editointioperaatiot jotka tuottavat arvon d_{mn}.

- Lävistäjän vaihto minimoivalla polulla aiheuttaa aina vähimmäistilakustannuksen Δ:

$$\Delta \quad = \min\{w(x \to y)/(||x| - |y||) \mid x \to y \in G, |x| \neq |y|\}.$$

- Arvoa d_{mn} laskettaessa voi siis olla korkeintaan d_{mn}/Δ lävistäjän vaihtoa.

- Tämän perusteella (d_{ij}):stä tarvitsee laskea vain tietty *vyö*, joka sisältää lävistäjät $0, \ldots, n - m$ ja ehkä joitakin muita lävistäjiä.

- Koska arvoa d_{mn} ei tunneta etukäteen, ei voida suoraan rajoittaa laskentaan tarvittavaa lävistäjien vyötä.

Tutkitaan aluksi tapausta, jossa on annettu kynnysarvo t etäisyyden arvolle:

- Halutaan tutkia, onko $d_{mn} \leq t$?

- Laskenta voidaan rajoittaa lävistäjien vyöhön $B_t : -x, -x+1, \ldots, n-m+x$, missä $x = \lfloor t/(2\Delta) - (n-m)/2 \rfloor$, koska jos minimipolku kulkee tämän ulkopuolelle, on sen kustannus vähintään $((n-m)+2(x+1))\Delta > t$.

- Simuloidaan Algoritmia 2.4.10 rajoitettuna vyöhön B_t. Olkoon d^t_{mn} saatu vastaus eli edullisimman vyön sisällä kulkevan polun kustannus,

- Jos $d^t_{mn} \leq t$, tiedetään, että $d_{mn} = d^t_{mn} \leq t$. Muuten $t < d_{mn} \leq d^t_{mn}$.

- Aikavaatimukseksi saadaan $O(||G||mt/\Delta)$.

Käytetään *tuplausalgoritmia* yleisen tapauksen ratkaisuun:

- Asetetaan ensin $t = t_{min} = (n - m)\Delta$ ja suoritetaan edellä oleva rajoitettu algoritmi. Saadaan tulos d^t_{mn}.

- Jos $d^t_{mn} > t$, tuplataan t:n arvo ja suoritetaan algoritmi uudelleen. Toistetaan tuplaus, kunnes $d^t_{mn} \leq t$, jolloin $d_{mn} = d^t_{mn}$.

- Olkoon t:n loppuarvo $t_{max} < 2d_{mn}$. Aikavaatimukseksi saadaan

$$\begin{aligned}&\mathcal{O}(||G||m(t_{min} + 2t_{min} + 4t_{min} + \cdots t_{max}/4 + t_{max}/2 + t_{max})/\Delta)\\&= \mathcal{O}(||G||mt_{max}/\Delta)\end{aligned}$$

Lause 2.4.11 *Yleinen editointietäisyys $d = D_G(A, B)$ voidaan laskea ajassa $O(||G||md/\Delta)$.*

Normaalin editointietäisyyden $D(A, B)$ tapauksessa $||G|| = O(1)$ ja $\Delta = 1$, joten lyhimmän kiertotien menettelyllä saadaan aikavaatimukseksi $O(dm)$, missä $d = D(A, B)$. Tämä on sama aikaraja kuin mikä saavutetaan lävistäjämenetelmällä. Lävistäjämenetelmä on kuitenkin käytännössä huomattavasti tehokkaampi (mutta ei yleisty yleisen editointietäisyyden laskentaan).

2.5 Pisin yhteinen alijono ja (poisto/lisäys)-editointietäisyys

Kiinnostava editointietäisyyden yksinkertaistus saadaan, kun editointioperaatiota M (muutos) ei sallita.

Määritelmä 2.5.1 *Merkkijonojen A ja B välinen (poisto/lisäys)-editointietäisyys $D'(A, B)$ on pienin editointioperaatioiden P (poisto) ja L (lisäys) määrä, joka tarvitaan muuntamaan A B:ksi.*

Huomataan seuraava yhteys kahden merkkijonon *pisimmän yhteisen alijonon* -ongelmaan.

Määritelmä 2.5.2 *Jono $C = c_1c_2 \cdots c_r$ on jonon $A = a_1a_2 \cdots a_m$* alijono*, jos C saadaan poistamalla jonosta A nolla tai useampia merkkejä. Jono C on jonojen A ja B* pisin yhteinen alijono *(*LCS*, longest common subsequence), jos C on pisin sellainen jono, että se on sekä A:n että B:n alijono.*

Esimerkki 2.5.3 *$LCS($`stockholm`, `tukholma`$)$ = `tkholm`; $D'($`stockholm`, `tukholma`$) = 5$:*

```
-tu--kholma
st-ockholm-
```

Lause 2.5.4 *a)* $D'(A,B) = |A| + |B| - 2 \cdot |LCS(A,B)|$.

b) Olkoot

(1) $a_{i_1} \to \epsilon, a_{i_2} \to \epsilon, \ldots, a_{i_p} \to \epsilon$ *ja*

(2) $\epsilon \to b_{j_1}, \epsilon \to b_{j_2}, \ldots, \epsilon \to b_{j_r}$

$D'(A,B)$:tä vastaavassa editointiaskeljonossa esiintyvät poistot ja lisäykset. Silloin $LCS(A,B) = C$, missä $C = A$, johon tehty poistot (1) ja $C = B$, johon tehty lisäyksiä (2) vastaavat poistot (jotka saadaan siis "kääntämällä" (2)).

Todistus.

b) Konstruktion perusteella C on selvästi A:n ja B:n yksikäsitteinen alijono. Jos C ei ole pisin mahdollinen, on $|C| < |C'|$, missä $C' = LCS(A,B)$. Mutta silloin voidaan $(|A| - |C'|)$:lla poistolla ja $(|B| - |C'|)$:lla lisäyksellä muuttaa A B:ksi, eli

$$D'(A,B) \leq |A| - |C'| + |B| - |C'| < |A| - |C| + |B| - |C| = D'(A,B),$$

mikä on ristiriita. Siis $C = LCS(A,B)$.

a) Kohdan b) nojalla $|LCS(A,B)| = |A| - p$ ja $|LCS(A,B)| = |B| - r$, joten

$$2 \cdot |LCS(A,B)| = |A| + |B| - (p+r) = |A| + |B| - D'(A,B).$$

□

Esimerkki 2.5.5 $|LCS(\texttt{stockholm}, \texttt{tukholma})| = |\texttt{tkholm}| = 6$; $D'(\texttt{stockholm}, \texttt{tukholma}) = 5 = 9 + 8 - 2 * 6$.

Aiempien algoritmien muunnokset $D'(A,B)$:lle.

- Etäisyyden $D'(A,B)$ laskemiseksi tulee matriisin (d_{ij}) palautuskaavan olla muodossa (alustus ei muutu):

$$d_{ij} = \min\{d_{i-1,j-1} + (\textbf{if } a_i = b_j \textbf{ then } 0 \textbf{ else } \infty), d_{i-1,j} + 1, d_{i,j-1} + 1\}$$

- Lävistäjälemma 2.3.1 pätee nyt muodossa $d_{ij} = d_{i-1,j-1}$ tai $d_{ij} = d_{i-1,j-1} + 2$.

- (poisto/lisäys)-editointietäisyys on yleisen editointietäisyyden erikoistapaus, jolla $\Delta = 1$.

- Aiemmin editointietäisyydelle $D(A, B)$ kehitettyihin algoritmeihin tulee vain pieniä muutoksia.

Lause 2.5.6 *$D'(A, B)$ ja $LCS(A, B)$ voidaan laskea perus dynaamisella ohjelmoinnilla ajassa $O(mn)$ sekä lävistäjämenetelmällä ja lyhimmän kiertotien menettelyllä ajassa $O(dm)$, missä $d = D'(A, B)$, $m = |A|$, $n = |B|$ ja $m \leq n$.*

2.6 Harva dynaaminen ohjelmointi

V. Mäkinen, G. Navarro & E. Ukkonen. Algorithms for transposition invariant string matching. Teoksessa *Proc. 20th International Symposium on Theoretical Aspects of Computer Science*, Springer-Verlag LNCS 2607, sivut 191–202, 2003.

- Harvassa dynaamisessa ohjelmoinnissa pyritään laskemaan vain osa dynaamisen ohjelmoinnin matriisista.

- LCS:n laskemiseksi on kehitetty useita harvaan dynaamiseen ohjelmointiin perustuvia menetelmiä. Näistä ensimmäinen on Hunt-Szymanski-algoritmi [4].

- Tarkastellaan hieman yksinkertaisempaa algoritmia, joka lisäksi yleistyy tavallisen editointietäisyyden laskentaan.

- Olkoon $M(A, B)$ täsmäävien merkkiparien joukko, eli $M(A, B) = M = \{(i, j) \mid a_i = b_j\}$.

- Ideana on laskea $d_{i,j} = D'(A_{1...i}, B_{1...j})$ vain pareille $(i, j) \in M$.

- Koska (m, n) ei välttämättä ole M:ssä, kuvitellaan, että A:n ja B:n lopussa on ylimääräinen merkki \$, eli $a_{m+1} = b_{n+1} = \$$, missä $\$ \notin \Sigma$. Lasketaankin $d_{m+1,n+1} = D'(A\$, B\$)$. Selvästi $(m + 1, n + 1) \in M = M(A\$, B\$)$ ja $D'(A, B) = D'(A\$, B\$) = d_{m+1,n+1}$.

Lause 2.6.1 *Arvoille* $d_{ij} = D'(A_{1\ldots i}, B_{1\ldots j})$, $(i,j) \in M$, *missä* $M = M(A\$, B\$) = \{(i,j) \mid a_i = b_j\} \cup \{(0,0)\}$, *pätee*

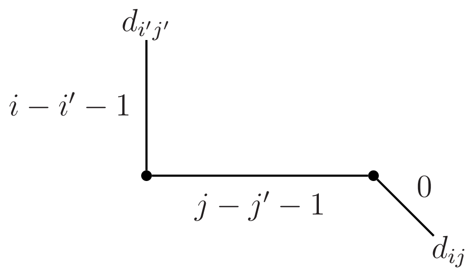
$$d_{ij} = \min\{d_{i',j'} + i - i' + j - j' - 2 \mid i' < i, j' < j, (i',j') \in M\}, \tag{2.6}$$

alustuksella $d_{0,0} = 0$. *Etäisyys* $D'(A,B) = d_{m+1,n+1}$.

Todistus.

- Tarkastellaan peräkkäisiä A:n merkkejä $a_{i'}$ ja a_i jonossa $LCS(A,B)$. Näitä vastaa merkit $b_{j'}$ ja b_j jonossa B. Vastaavassa $D'(A,B)$:n editointioperaatiojonossa identiteettioperaatioiden $a_{i'} \rightarrow b_{j'}$ ja $a_i \rightarrow b_j$ välissä on vain poistoja ja lisäyksiä.

- Pienin kustannus muuntaa $A_{i'+1\ldots i-1}$ jonoksi $B_{j'+1\ldots j-1}$ poistoilla ja lisäyksillä on $(i - i' - 1) + (j - j' - 1) = i - i' + j - j' - 2$.



- Palautuskaavassa tutkitaan kullekin parille (i,j), $a_i = b_j$, kaikki mahdolliset edeltävät parit (i',j'), $a_{i'} = b_{j'}$; näistä valitaan se josta saadaan pienin yhteiskustannus muuntamaan ensin $A_{1\ldots i'}$ jonoksi $B_{1\ldots j'}$ (kustannus $d_{i',j'}$), ja sitten jono $A_{i'\ldots i}$ jonoksi $B_{j'\ldots j}$ (kustannus $i - i' + j - j' - 2$).

- Alustus toimii oikein; jos $a_i \rightarrow b_j$ on ensimmäinen identiteettioperaatio editointijonossa, maksaa $A_{1\ldots i-1}$:n muuntaminen $B_{1\ldots j-1}$:ksi $i - 1 + j - 1 = d_{0,0} + i - 0 + j - 0 - 2 = d_{i,j}$.

- Tehdään induktio-oletus, että kaikki arvot $d_{i',j'}$, $i' < i$, $j' < j$, on laskettu oikein. Tästä ja edelläolevasta palautuskaavan tulkinnasta seuraa, että d_{ij} saa tällöin oikean arvon.

$\square$

Algoritmin johtaminen palautuskaavasta.

- Kirjoitetaan (2.6) muotoon

$$d_{ij} \;\; = i + j - 2 + \min\{d_{i',j'} - i' - j' \mid i' < i, j' < j, (i',j') \in M\},$$

tuomalla ulos termit jotka eivät riipu i':sta ja j':sta.

- Lasketaan arvot d_{ij} *käänteisessä sarakejärjestyksessä* $<^{ks}$:

$$(i',j') \;\; <^{ks} (i,j) \text{ jos ja vain jos } j' < j \text{ tai } (j' = j \text{ ja } i' > i).$$

- Olkoon $M_{ij} = \{(i',j') \in M \mid (i',j') <^{ks} (i,j)\}$ ne M:n alkiot, jotka käsitellään ennen (i,j):tä.

  Tällöin palautuskaava voidaan kirjoittaa muotoon

$$d_{ij} = i + j - 2 + \min\{d_{i',j'} - i' - j' \mid i' < i, (i',j') \in M_{ij}\}, \qquad (2.7)$$

koska

$$i' < i \text{ ja } j' < j \text{ jos ja vain jos } i' < i \text{ ja } (i',j') <^{ks} (i,j).$$

- *Algoritmin idea:*
  - Käsitellään parit $(i',j') \in M$ käänteisessä sarakejärjestyksessä.
  - Talletetaan kukin arvo $d_{i',j'} - i' - j'$ tietorakenteeseen $\mathcal{T}$ (määritellään alla) hakuavaimena i' siten, että $\mathcal{T}$:stä voi hakea minimiarvoa niiden arvojen joukosta joissa rivinumero $i' < i$.
  - Olkoon tämä minimi d.
  - Tällöin $d_{ij} = i + j - 2 + d$.

Lemma 2.6.2 *Prioriteettihakupuulla (priority search tree [7]) $\mathcal{T}$ seuraavat operaatiot voidaan toteuttaa ajassa $O(\log n)$, missä n on talletettujen alkioiden lukumäärä:*

$Insert(i,v)$ **:** *Lisää parin (i,v) puuhun hakuavaimena i. Jos avain i on jo puussa, korvataan siihen liitetty arvo v' arvolla $\min(v,v')$.*

$v = Minimum(l,r)$ **:** *Palauttaa pienimmän arvon v, jolla on puussa on pari (i,v), $l \le i \le r$.*

Todistus. Prioriteettihakupuita käsitellään tarkemmin kurssilla *geometriset menetelmät*. Hahmotellaan tässä vain idea:

- Prioriteettihakupuu on lähes normaali tasapainotettu binäärihakupuu.
- Ainoa ero on, että kunkin alipuun juureen on talletettu kyseisen alipuun pienin v-arvo. Näiden minimien päivittäminen normaalin lisäyksen yhteydessä on helppoa.
- Millä tahansa osavälillä $[l, r]$ on olemassa $\mathcal{O}(\log n)$ alipuuta, jotka sisältävät täsmälleen ne alkiot, joiden avain on kyseisellä välillä. Näiden alipuiden juurisolmut voidaan löytää $\mathcal{O}(\log n)$ ajassa, ja pienin niihin talletetuista v-arvoista on $Minimum(l, r)$.

$\square$

Algoritmi 2.6.3 *$D'(A, B)$:n laskenta harvalla dynaamisella ohjelmoinnilla*

Syöte: Jonot A ja B, $|A| = m \leq n = |B|$, sekä
joukko $M = \{(i, j) \mid a_i = b_j\}$ käänteisessä sarakejärjestyksessä
Tuloste: $D'(A, B)$
$\mathcal{T}.Insert(0, 0)$; (* lisätään $d_{0,0} = 0$ avaimena rivi 0 *)
for $p := 1$ **to** $|M|$ **do**
  $(i, j) := M[p]$;
  $d := i + j - 2 + \mathcal{T}.Minimum(-\infty, i - 1)$; (* $d = d_{i,j}$ *)
  $\mathcal{T}.Insert(i, d - i - j)$;
return $\mathcal{T}.Minimum(-\infty, m) + m + n$; (* $d_{m+1,n+1}$ *)

Huom: Lisämerkkiä \$ ei siis oikeasti tarvittu.

Lause 2.6.4 *Etäisyys $D'(A, B)$ tai $|LCS(A, B)|$ voidaan laskea Algoritmilla 2.6.3 ajassa $O((n+|M|) \log m)$ ja tilassa $O(|M|)$, missä $M = \{(i, j) \mid a_i = b_j\}$.*

Todistus.

- Algoritmin oikeellisuus seuraa edellä johdetuista palautuskaavoista; käänteinen sarakejärjestys takaa, että kutsu $\mathcal{T}.Minimum(-\infty, i - 1)$ vastaa minimin ottoa kaavassa (2.7).
- Ainoa ero on, että $\mathcal{T}$:ssä säilytetään vain pienin arvo kullakin rivillä i'. Saman rivinumeron omaavat arvot kuuluvat kuitenkin aina samoihin hakualueisiin, joten kaikkiin hakuihin vastataan edelleen oikein.
- Algoritmi kutsuu $O(|M|)$ kertaa puun $\mathcal{T}$ operaatioita. Kukin näistä vie $O(\log m)$ ajan, koska puussa voi olla vain $m + 1$ alkiota kerrallaan.

- Lopuksi, joukko M voidaan muodostaa helposti ajassa $O(|\Sigma| + m + n + |M|)$ vakioaakkostolla, tai yleisessä tapauksessa ajassa $O((m + n)\log m + |M|)$. Koska $m \leq n$, saadaan väitetty yläraja.

$\square$

Huomautus 2.6.5 *Edellisen lauseen tulosta voidaan vielä parantaa; saadaan aikavaativuuden ylärajaksi $O(n \log m + |M| \log\log m)$ ja tilaksi $O(m)$. Aikavaativuuden parannus perustuu prioriteettihakupuun korvaamiseen tehokkaammalla tietorakenteella. Tilavaatimusta voidaan pienentää muodostamalla M sarakkeittain samalla kun suoritetaan Algoritmia 2.6.3.*

2.7 Hahmon likimääräisten esiintymien etsiminen

G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys* 33(1):31–88, 2001.

- Olkoon $S = s_1 s_2 \cdots s_n \in \Sigma^*$ *teksti* ja $P = p_1 p_2 \cdots p_m \in \Sigma^*$ *hahmo*.

- Olkoon k jokin vakio, $0 \leq k \leq m$.

Määritelmä 2.7.1 *Hahmon P* likimääräinen esiintymä k virheellä *(k differences) on tekstin S osajono X, jolla $D(P, X) \leq k$.*

- Tässä $D(\cdot, \cdot)$ on normaali editointietäisyys (operaatiot M, L ja P, joiden kustannus on 1) mutta määritelmä yleistyy myös muille muille editointietäisyyden muunnelmille.

- Käytännössä usein halutaan vain kaikki esiintymien loppukohdat. Kuhunkin loppukohtaan voi päättyä useita esiintymiä.

Dynaaminen ohjelmointi

- Määritellään $g_{ij} = \min\{D(p_1 \cdots p_i, s_{j-\ell+1} \cdots s_j) \mid 0 \leq \ell \leq j\}$, kaikilla $0 \leq i \leq m$, $0 \leq j \leq n$.
- Tällöin j on esiintymän loppukohta, jos ja vain jos $g_{mj} \leq k$.

g_{ij}		r	e	m	a	c	h	i	n	e
	0	0	0	0	0	0	0	0	0	0
m	1	1	1	0	1	1	1	1	1	1
a	2	2	2	1	0	1	2	2	2	2
t	3	3	3	2	1	1	2	3	3	3
c	4	4	4	3	2	1	2	3	4	4
h	5	5	5	4	3	2	1	2	3	4

Lause 2.7.2 *Arvot g_{ij}, $0 \leq i \leq m, 0 \leq j \leq n$ voidaan laskea palautuskaavalla:*

$$
\begin{aligned}
g_{0j} &= 0, \qquad 0 \leq j \leq n, \\
g_{i0} &= i, \qquad 1 \leq i \leq m, \text{ ja} \\
g_{ij} &= \min\{g_{i-1,j-1} + (\textbf{\textit{if}}\ p_i = s_j\ \textbf{\textit{then}}\ 0\ \textbf{\textit{else}}\ 1), g_{i-1,j} + 1, g_{i,j-1} + 1\}.
\end{aligned}
$$

Todistus. Induktio.
1. Alustus on oikein:

$$
\begin{aligned}
g_{00} &= D(\epsilon, \epsilon) = 0, \\
g_{0j} &= D(\epsilon, s_j - 0 + 1 \cdots s_j) = D(\epsilon, \epsilon) = 0, \\
g_{i0} &= D(p_1 \cdots p_i, \epsilon) = i.
\end{aligned}
$$

2. Induktioaskel: Oletetaan, että $g_{i',j'}$ on oikein, kun $i' + j' < i + j$. Merkitään $d_{ij}^{\ell} = D(p_1 \cdots p_i, s_j - \ell + 1 \cdots j)$. Tällöin

$$
\begin{aligned}
g_{ij} &= \min_{0 \leq \ell \leq j} d_{ij}^{\ell} \\
&= \min_{\ell} \min\{d_{i-1,j-1}^{\ell} + (\textbf{if}\ p_i = s_j\ \textbf{then}\ 0\ \textbf{else}\ 1), d_{i-1,j}^{\ell} + 1, d_{i,j-1}^{\ell} + 1\} \\
&= \min\{\min_{\ell} d_{i-1,j-1}^{\ell} + (\textbf{if}\ p_i = s_j\ \textbf{then}\ 0\ \textbf{else}\ 1), \min_{\ell} d_{i-1,j}^{\ell} + 1, \min_{\ell} d_{i,j-1}^{\ell} + 1\} \\
&= \min\{g_{i-1,j-1} + (\textbf{if}\ p_i = s_j\ \textbf{then}\ 0\ \textbf{else}\ 1), g_{i-1,j} + 1, g_{i,j-1} + 1\}.
\end{aligned}
$$

□

Algoritmi 2.7.3 *Likimääräinen hahmon sovitus, k virhettä*

Syöte: P, S, k.
Tuloste: P:n likimääräiset esiintymäkohdat k virheellä S:ssä
(1) **for** $i = 0$ **to** m **do** $g_{i0} := i$;
(2) **for** $j = 1$ **to** n **do**
(3) $g_{0j} := 0$;
(4) **for** $i := 1$ **to** m **do**
(5) $g_{ij} := \min\{g_{i-1,j-1}+$ (**if** $p_i = s_j$ **then** 0 **else** 1), $g_{i-1,j}+1$, $g_{i,j-1}+1\}$;
(6) **if** $g_{mj} \leq k$ **then** write(j);

- Algoritmin 2.7.3 aikavaatimus on $\mathcal{O}(mn)$ ja tilavaatimus $\mathcal{O}(mn)$.
- Tilavaatimus voidaan pienentää $\mathcal{O}(m)$:ään pitämällä vain yksi sarake muistissa kuten algoritmissa 2.2.4.

Lause 2.7.4 *Likimääräisen hahmonsovitus k virheellä -ongelma voidaan ratkaista Algoritmilla 2.7.3 ajassa $O(mn)$ ja tilassa $O(m)$.*

- Algoritmi voidaan muuttaa tulostamaan kullekin loppukohdalle myös siihen päättyvän esiintymän alkukohta samassa ajassa ja tilassa.
- Jos $g_{ij} = d$, niin $D(p_1 \cdots p_i, s_t \cdots s_j) = d$ jollakin $t = t_{ij}$. Talletetaan t_{ij} yhdessä g_{ij}:n kanssa.
- t_{ij} on helppo laskea minimoinnin yhteydessä. Esimerkiksi, jos $g_{ij} = g_{i,j-1} + 1$, niin $t_{ij} = t_{i,j-1}$.
- Kaikkia esiintymiä ei kuitenkaan näin löydetä. Ensinnäkin, t_{ij} ei ole välttämättä yksikäsitteinen. Toiseksi, jos $g_{mj} < d \leq k$, algoritmi ei löydä j:hin päättyviä esiintymiä, joiden editointietäisyys hahmosta on d.

Lävistäjälemma pätee nytkin:

Lemma 2.7.5 *Jokaisella g_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$: $g_{ij} = g_{i-1,j-1}$ tai $g_{ij} = g_{i-1,j-1} + 1$.*

Tämän ominaisuuden avulla (g_{ij}):n laskentaa voidaan rajoittaa k virhettä -versiossa:

- Pidetään kirjaa siitä, missä kullakin sarakkeella $g_{*,j}$ on viimeinen g_{ij}, joka on korkeintaan k. Silloin sarakkeella $g_{*,j+1}$ voi olla arvoja, jotka ovat korkeintaan k, vain riveillä $0, \ldots, i+1$. Niitä pitemmälle ei kannata laskea.
- Merkataan $top[j] = \max\{i \mid i \leq m \text{ ja } g_{i,j} = k\}$. Tällöin $top[j+1] \leq top[j] + 1$.

Algoritmi 2.7.6 *Likimääräinen hahmon sovitus, tehostettu versio*

Syöte: P, S, k.
Tuloste: P:n likimääräiset esiintymäkohdat k virheellä S:ssä.

for $i = 0$ **to** $\min(k+1, m)$ **do** $g_{i0} := i$;
$top := \min(k+1, m)$; (* $top[0] = \min(k, m)$ *)
for $j = 1$ **to** n **do**
    $g_{0j} := 0$;
    for $i := 1$ **to** top **do**
        $g_{ij} := \min\{g_{i-1,j-1} +$ (**if** $p_i = s_j$ **then** 0 **else** 1), $g_{i-1,j} + 1$, $g_{i,j-1} + 1\}$;
    while $g_{top,j} > k$ **do** $top := top - 1$; (* löydetään $top[j]$ *)
    if top = m **then** $write(j)$;
    else
        $top := top + 1$;
        $g_{top,j} := \infty$; (* alustus seuraavaa saraketta varten *)

$k = 1$

g_{ij}		r	e	m	a	c	h	i	n	e
	0	0	0	0	0	0	0	0	0	0
m	1	1	1	0	1	1	1	1	1	1
a	2	2	2	1	0	1	2	2	2	2
t	3	3	3	2	1	1	2	3	3	3
c	4	4	4	3	2	1	2	3	4	4
h	5	5	5	4	3	2	1	2	3	4

- Algoritmin 2.7.6 aikavaatimus on verrannollinen lasketun (g_{ij}):n osan pinta-alaan.
- Jos S ja P ovat satunnaisia, voidaan osoittaa, että keskimääräinen aikavaatimus on $\mathcal{O}(kn)$.
- Tilavaatimus $\mathcal{O}(m)$ saavutetaan pitämällä vain yksi sarake muistissa.

Lause 2.7.7 *Likimääräinen hahmonsovitus k virheellä -ongelma voidaan ratkaista Algoritmilla 2.7.6 keskimäärin ajassa $O(kn)$ ja tilassa $O(m)$.*

Todistus. Katso Chang & Lampe [1]. □

- Algoritmi 2.7.6 toimii hyvin käytännössä, jos k on pieni.
- Algoritmi 2.7.6 toimii myös muille etäisyysmitoille, joilla lävistäjät ovat monotonisia (arvot eivät koskaan laske), esim. (poisto/lisäys)-editointietäisyys.
- Chang & Lampe [1] esittävät toisenlaisen algoritmin, joka toimii keskimäärin ajassa $O(kn)$.
- Lävistäjälemmaan perustuen on kehitetty useita likimääräisen hahmonsovituksen algoritmeja, joiden aikavaatimus on $O(kn)$ pahimmassakin tapauksessa (eikä vain keskimäärin), esim. Landau & Vishkin [5].

2.7.1 Myersin bittirinnakkainen algoritmi

G. Myers. A fast bit-vector algorithm for approximate pattern matching based on dynamic programming. *JACM* 46(3):395–415, 1999.

- Nopeutetaan matriisin (g_{ij}) laskentaa soveltamalla bittirinnakkaisuutta. Rajoitumme lyhyisiin hahmoihin, joilla matriisin sarake voidaan käsitellä yhdessä sanassa.
- Matriisi (g_{ij}) esitetään muodossa, jossa talletetaan solujen arvojen sijasta kahden vierekkäisen solun erotus.
- Määritellään *vaakaero* (horizontal delta) $\Delta h_{ij} = g_{ij} - g_{i,j-1}$ ja *pystyero* (vertical delta) $\Delta v_{ij} = g_{ij} - g_{i-1,j}$.
- Koska $g_{i0} = i$ ja $g_{0j} = 0$, pätee

$$\begin{aligned} g_{ij} &= \Delta v_{1j} + \Delta v_{2j} + \cdots + \Delta v_{ij} \\ g_{ij} &= i + \Delta h_{1j} + \Delta h_{2j} + \cdots + \Delta h_{ij} \end{aligned}$$

- Kaikki arvot Δh_{ij} ja Δv_{ij} voidaan esittää kahdella bitillä, mikä seuraa Lemmasta 2.7.8.

Lemma 2.7.8 *Kaikilla i, j on voimassa $\Delta h_{ij}, \Delta v_{ij} \in \{-1, 0, 1\}$.*