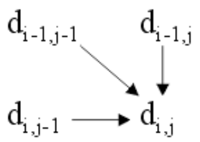
- Suoraviivaisen ratkaisun tilavaatimus on $\Theta(mn)$ (eli matriisin (d_{ij}) vaatima tila).

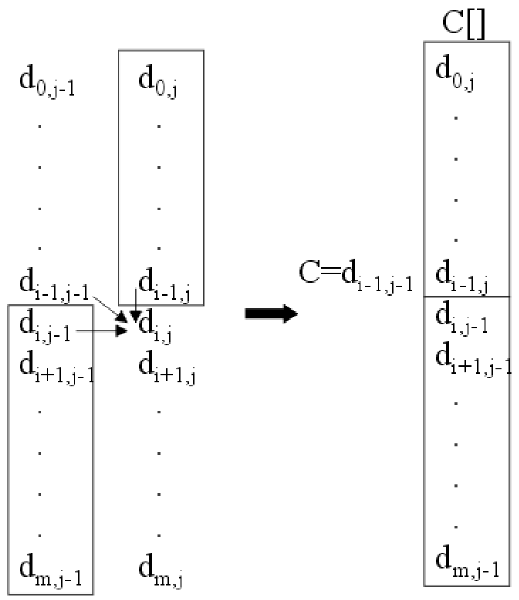
Tilavaatimus tarkemmin:

- d_{ij}:t riippuvat toisistaan seuraavasti:



- Sarakkeen j arvojen d_{ij}, $1 \leq i \leq m$, laskemiseksi riittää tuntea edellisen sarakkeen $j-1$ arvot $d_{i,j-1}$, $1 \leq i \leq m$. Alustuksen perusteella tiedetään $d_{0j} = j$, josta alaspäin laskenta sujuu edellisen sarakkeen arvojen avulla.

- Tarvitaan työtilaksi taulu $C[0 \ldots m]$ ja yksi apumuuttuja C, jotka esittävät vuorossa olevan sarakkeen jo laskettua osaa ja edellisen sarakkeen vielä tarvittavaa osaa:



- $C[0 \ldots i-1]$ on sarakkeen j jo laskettu osuus.

- C ja $C[i \ldots m]$ muodostavat sarakkeen $j-1$ vielä tarvittavan osuuden.

Algoritmi 2.2.4 *$D(A, B)$:n laskenta tilassa $\Theta(m)$.*

Syöte: $A = a_1 a_2 \cdots a_m$, $B = b_1 b_2 \cdots b_n$
Tuloste: $d_{mn} = D(A, B)$
Huom. Kaikki matriisin (d_{ij}) alkiot lasketaan, mutta niitä ei talleteta.

for $i := 0$ **to** m **do** $C[i] := i$ (* alustus $j = 0$*)
for $j := 1$ **to** n **do**
    $C := C[0]; C[0] := j$ (* alustus $(i = 0)$ *)
    for $i := 1$ to m **do**
    (* invariantti: $C = d_{i-1,j-1}$, $C[i-1] = d_{i-1,j}$, $C[i] = d_{i,j-1}$ *)
        $tmp := \min(C + (\textbf{if } a_i = b_j \textbf{ then } 0 \text{ else } 1), C[i-1]+1, C[i]+1)$
        $C := C[i]$ (* $d_{i,j-1}$ talteen *)
        $C[i] := tmp$
write $C[m]$

- Aikavaatimus on $\Theta(mn)$, koska $C[0 \ldots m]$ täytetään n kertaa. Tilavaatimus on $\Theta(m)$.
- Huomaa, että koska $D(A, B) = D(B, A)$, voidaan Alg. 2.2.4:ssa aina valita jonoksi A lyhyempi verrattavista jonoista, joten $m \leq n$ pätee aina.

Editointietäisyyttä vastaavan operaatiojonon jäljittäminen

- Kuten yleensäkin dynaamisessa ohjelmoinnissa, voidaan haetun optimiratkaisun (tässä: d_{mn}) antavat alkeisaskeleet (tässä: editointioperaatiot) selvittää jälkikäteen, kun tulos on laskettu. Tuloksesta d_{mn} peruutetaan kohti alkupistettä d_{00} ja tutkitaan mitä reittiä pitkin minimointi eteni (d_{ij}):ssä.
- Tämän helpottamiseksi voidaan d_{ij}:hin liittää tieto siitä, mistä alkiosta ($d_{i-1,j-1}$, $d_{i-1,j}$ tai $d_{i,j-1}$) siihen päästiin minimoinnissa. Alkio d_{ij} on
  (1) $d_{i-1,j-1} + (\textbf{if } a_i = b_j \textbf{ then } 0 \textbf{ else } 1)$ tai
  (2) $d_{i-1,j} + 1$ tai
  (3) $d_{i,j-1} + 1$.
- Vastaavat operaatiot ovat

(1) muutos tai täsmäys: $a_i \to b_j$,

(2) poisto: $a_i \to \epsilon$ ja

(3) lisäys: $\epsilon \to b_j$.

- Ratkaisun jäljittäminen:
  1. Kun d_{ij} on laskettu (Algoritmissa 2.2.3), talleta kaikilla i, j joukkoon $pred_{ij}$
     - alkio $(i-1, j-1)$, jos (1) pätee,
     - alkio $(i-1, j)$, jos (2) pätee tai
     - alkio $(i, j-1)$, jos (3) pätee.

  2. Kun d_{mn} on laskettu, seuraa *pred*-linkkejä d_{00}:aan.

Esimerkki 2.2.5

d		h	a	n	d	b	a	l	l
	0	1	2	3	4	5	6	7	8
b	1	1	2	3	4	4	5	6	7
a	2	2	1	2	3	4	4	5	6
l	3	3	2	2	3	4	5	4	5
l	4	4	3	3	3	4	5	5	4
a	5	5	4	4	4	4	4	5	5
d	6	6	5	5	4	5	5	5	6

Löydetään 7 ratkaisua (polkua d_{mn}:stä d_{00}:aan), esim.:

```
LLLLTTTTPP      MTLMMTLM      MTMMLTML
----ballad      ba-lla-d      ball-ad-
handball--      handball      handball
```

- Ratkaisuja voi olla siis useita. Sovelluksesta riipuen näillä saattaa vielä olla jokin arvojärjestys.

- Koska Alg. 2.2.4 ei talleta matriisia (d_{ij}), sitä käytettäessä ratkaisua ei voida jäljittää. Usein kuitenkin pelkkä d_{mn}-arvo riittää.

2.3 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 kokonaisluku, $0 \leq k \leq m$.

Määritelmä 2.3.1 *Hahmon P likimääräinen esiintymä k virheellä (k differences) on tekstin S osajono X, jolla $D(P, X) \leq k$.*

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

2.3.1 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 2.3.2 *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:

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

2. Induktioaskel: Kuten lauseessa 2.2.1. □

Algoritmi 2.3.3 *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) $\quad g_{0j} := 0$
(4) $\quad$ **for** $i := 1$ **to** m **do**
(5) $\qquad g_{ij} := \min\{g_{i-1,j-1} +$ (**if** $p_i = s_j$ **then** 0 **else** 1), $g_{i-1,j} + 1$, $g_{i,j-1} + 1\}$
(6) $\quad$ **if** $g_{mj} \leq k$ **then** write(j)

- Algoritmin 2.3.3 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 2.3.4 *Likimääräisen hahmonsovitus k virheellä -ongelma voidaan ratkaista Algoritmilla 2.3.3 ajassa $\mathcal{O}(mn)$ ja tilassa $\mathcal{O}(m)$.*

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.

2.3.2 Rajoitettu dynaaminen ohjelmointi

E. Ukkonen: Finding approximate pattern in strings. *J. Algorithms* 6:132–137, 1985.

- Matriisin (g_{ij}) rakenteessa on säännönmukaisuus, joka johtaa perusalgoritmeja tehokkaampaan ratkaisuun.
- Matriisin (g_{ij}) *lävistäjään* k, $-m \leq k \leq n$, kuuluvat ne alkiot g_{ij} joilla $j - i = k$.

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

- Matriisin (g_{ij}) alkioiden arvot voivat ainoastaan pysyä ennallaan tai kasvaa, kun edetään jotakin lävistäjää pitkin. Kun arvo kasvaa, lisäys on aina yksi.
- Editointietäisyyden laskemisessa käytettävällä matriisilla (d_{ij}) on sama ominaisuus.

Lemma 2.3.5 *Jokaisella g_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$: $g_{ij} = g_{i-1,j-1}$ tai $g_{ij} = g_{i-1,j-1} + 1$. (Huomaa, että g_{ij} ja $g_{i-1,j-1}$ ovat samalla lävistäjällä.)*

Todistus. Koska g_{ij} on kokonaisluku, ja koska määritelmän mukaan $g_{ij} \leq g_{i-1,j-1} + 1$, riittää osoittaa, että

$$g_{i-1,j-1} \quad \leq \quad g_{ij}.$$

Tämä todistetaan induktiolla $i + j$:n suhteen.
Ainakin yksi seuraavista pätee:

(a) $g_{ij} = g_{i-1,j-1} + (\textbf{if } p_i = s_j \textbf{ then } 0 \textbf{ else } 1)$. Tällöin $g_{ij} \geq g_{i-1,j-1}$.

(b) $i = 1$. Tällöin $g_{0,j-1} = 0 \leq g_{1j}$.

(c) $i > 1$ ja $g_{ij} = g_{i-1,j} + 1$. Tällöin pätee

$$g_{ij} = g_{i-1,j} + 1 \geq g_{i-2,j-1} + 1 \geq g_{i-1,j-1}$$

induktio-oletuksen (ensimmäinen $\geq$) ja määritelmän (toinen $\geq$) nojalla. Huomaa, että induktio-oletusta voitiin käyttää, koska $i > 1$.

(d) $g_{ij} = g_{i,j-1} + 1$. Tämä voi tapahtua vain, jos $j > 1$, koska $g_{i1} \leq g_{0,j-1} + 1 < g_{i0} + 1$. Tällöin pätee

$$g_{ij} = g_{i,j-1} + 1 \geq g_{i-1,j-2} + 1 \geq g_{i-1,j-1}\ .$$

□

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

- 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 2.3.6 *Likimääräinen hahmon sovitus, rajoitettu 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 2.3.6 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 2.3.7 *Likimääräinen hahmonsovitus k virheellä -ongelma voidaan ratkaista Algoritmilla 2.3.6 keskimäärin ajassa $O(kn)$ ja tilassa $O(m)$.*

Todistus. Katso

W. Chang & J. Lampe: Theoretical and empirical comparisons of approximate string matching algorithms. Proc. 3rd Symposium on Combinatorial Pattern Matching (CPM), LNCS 644, Springer, 1992, 175–184.

□

- Algoritmi 2.3.6 toimii hyvin käytännössä, jos k on pieni.

- Chang & Lampe 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.

G. Landau & U. Vishkin: Fast string matching with k differences. *J. Comput. Syst. Sci.* 37:63–78, 1988.

2.4 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 talletetaaan 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

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

- Kaikki arvot Δh_{ij} ja Δv_{ij} voidaan esittää kahdella bitillä, mikä seuraa Lemmasta 2.4.1.

Lemma 2.4.1 *Kaikilla i, j on voimassa $\Delta h_{ij}, \Delta v_{ij} \in \{-1, 0, 1\}$.*

- Laskenta etenee sarakkeittain

$$\Delta v_{*0} \rightarrow \Delta h_{*1} \rightarrow \Delta v_{*1} \rightarrow \Delta h_{*2} \rightarrow \cdots$$

- Lisäksi lasketaan viimeisen rivin arvot $sc_j = g_{mj}$, $0 \leq j \leq n$, kaavoilla $sc_0 = m$ ja $sc_j = sc_{j-1} + \Delta h_{mj}$.

Esimerkki 2.4.2 *Vaaka- ja pystyerot hahmolle* `match` *ja tekstille* `remachine` *(-1 ja 1 on korvattu symboleilla $-$ ja $+$).*

			r		e		m		a		c		h		i		n		e
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	+		+		+		0		+		+		+		+		+		+
	1	0	1	0	1	−	0	+	1	0	1	0	1	0	1	0	1	0	1
a	+		+		+		+		−		0		+		+		+		+
	2	0	2	0	2	−	1	−	0	+	1	+	2	0	2	0	2	0	2
t	+		+		+		+		+		0		0		+		+		+
	3	0	3	0	3	−	2	−	1	0	1	+	2	+	3	0	3	0	3
c	+		+		+		+		+		0		0		0		+		+
	4	0	4	0	4	−	3	−	2	−	1	+	2	+	3	+	4	0	4
h	+		+		+		+		+		+		−		−		−		0
	5	0	5	0	5	−	4	−	3	−	2	−	1	+	2	+	3	+	4

- Dynaamisen ohjelmoinnin matriisin solun g_{ij} laskemiseen tarvitaan soluja $g_{i-1,j}, g_{i-1,j-1}$ ja $g_{i,j-1}$. Näihin liittyy kaksi pystyeroa $\Delta v^{\text{in}} = \Delta v_{i,j-1}$ ja $\Delta v^{\text{out}} = \Delta v_{i,j}$ sekä kaksi vaakaeroa $\Delta h^{\text{in}} = \Delta h_{i,j-1}$ ja $\Delta h^{\text{out}} = \Delta h_{i,j}$.

- Todistuksessa (mutta ei laskennassa) käytetään lisäksi arvoa $\Delta d = g_{ij} - g_{i-1,j-1}$.

$$
\begin{tikzcd}
g_{i-1,j-1} \arrow[r, "\Delta h^{\text{in}}"] \arrow[d, "\Delta v^{\text{in}}"'] \arrow[dr, "\Delta d"] & g_{i-1,j} \arrow[d, "\Delta v^{\text{out}}"] \\
g_{i,j-1} \arrow[r, "\Delta h^{\text{out}}"'] & g_{ij}
\end{tikzcd}
$$

- Lisäksi määritellään, että $Eq = Eq_{ij}$ on 1, jos $p_i = s_j$, muutoin 0.

- Kuvataan laskenta funktiona, jonka argumentteina ovat $\Delta v^{\text{in}}, \Delta h^{\text{in}}$ ja Eq sekä tuloksina Δd, Δv^{out} ja Δh^{out}.

Lemma 2.4.3 *Jos $Eq = 1$ tai $\Delta v^{\text{in}} = -1$ tai $\Delta h^{\text{in}} = -1$,*
niin pätee $\Delta d = 0$, $\Delta v^{\text{out}} = -\Delta h^{\text{in}}$ ja $\Delta h^{\text{out}} = -\Delta v^{\text{in}}$.
Muuten $\Delta d = 1$, $\Delta v^{\text{out}} = 1 - \Delta h^{\text{in}}$ ja $\Delta h^{\text{out}} = 1 - \Delta v^{\text{in}}$.

Todistus.

- g_{ij}:n palautuskaava voidaan kirjoittaa muotoon:

$$\begin{aligned} g_{ij} &= \min\{g_{i-1,j-1} + (\textbf{if } p_i = s_j \textbf{ then } 0 \textbf{ else } 1), g_{i,j-1}+1, g_{i-1,j}+1\} \\ &= g_{i-1,j-1} + \min\{1 - Eq, \Delta v^{\text{in}} + 1, \Delta h^{\text{in}} + 1\}. \end{aligned}$$

- Merkitään $\Delta d = g_{ij} - g_{i-1,j-1} = \min\{1 - Eq, \Delta v^{\text{in}} + 1, \Delta h^{\text{in}} + 1\}$. Siis $\Delta d = 0$, jos $Eq = 1$ tai $\Delta v^{\text{in}} = -1$ tai $\Delta h^{\text{in}} = -1$. Muuten $\Delta d = 1$.

- Selvästi $\Delta d = \Delta v^{\text{in}} + \Delta h^{\text{out}} = \Delta h^{\text{in}} + \Delta v^{\text{out}}$, mistä saadaan $\Delta v^{\text{out}} = \Delta d - \Delta h^{\text{in}}$ ja $\Delta h^{\text{out}} = \Delta d - \Delta v^{\text{in}}$.

□

Arvot Eq_{ij} ja Δd ovat bittejä mutta Δv_{ij} ja Δh_{ij} ovat "trittejä". Siirrytään nyt laskemaan kokonaan biteillä. Samalla siirrytään aritmeettisista loogisiin operaatioihin tulkinnalla 0 on epätosi ja 1 on tosi.

- Esitetään Δv_{ij} kahdella bitillä Pv_{ij} ja Mv_{ij}, joista edellinen on 1, jos $\Delta v_{ij} = 1$, ja vastaavasti jälkimmäinen on 1, jos $\Delta v_{ij} = -1$. Esitetään Δh_{ij} vastaavasti kahdella bitillä Ph_{ij} ja Mh_{ij}. Tällöin

$$\begin{aligned} \Delta v_{ij} &= Pv_{ij} - Mv_{ij} \\ \Delta h_{ij} &= Ph_{ij} - Mh_{ij} \end{aligned}$$

Lemma 2.4.4

$$\begin{aligned} Pv^{\text{out}} &= Mh^{\text{in}} \vee \neg(Xv \vee Ph^{\text{in}}) \\ Mv^{\text{out}} &= Ph^{\text{in}} \wedge Xv \\ Ph^{\text{out}} &= Mv^{\text{in}} \vee \neg(Xh \vee Pv^{\text{in}}) \\ Mh^{\text{out}} &= Pv^{\text{in}} \wedge Xh \end{aligned}$$

missä $Xv = Eq \vee Mv^{\text{in}}$ *ja* $Xh = Eq \vee Mh^{\text{in}}$.

Todistus. Symmetrian vuoksi riittää osoittaa väite Pv:lle ja Mv:lle.

- Lemman 2.4.3 mukaan

$$\begin{aligned} Pv^{\text{out}} &= (\neg\Delta d \wedge Mh^{\text{in}}) \vee (\Delta d \wedge \neg Ph^{\text{in}}) \\ Mv^{\text{out}} &= (\neg\Delta d \wedge Ph^{\text{in}}) \vee (\Delta d \wedge 0) = \neg\Delta d \wedge Ph^{\text{in}} \end{aligned}$$

- Koska $\Delta d = \neg(Eq \vee Mv^{\mathrm{in}} \vee Mh^{\mathrm{in}}) = \neg(Xv \vee Mh^{\mathrm{in}}) = \neg Xv \wedge \neg Mh^{\mathrm{in}}$, pätee

$$\begin{aligned}
Pv^{\mathrm{out}} &= ((Xv \vee Mh^{\mathrm{in}}) \wedge Mh^{\mathrm{in}}) \vee (\neg Xv \wedge \neg Mh^{\mathrm{in}} \wedge \neg Ph^{\mathrm{in}}) \\
&= Mh^{\mathrm{in}} \vee \neg(Xv \vee Mh^{\mathrm{in}} \vee Ph^{\mathrm{in}}) \\
&= Mh^{\mathrm{in}} \vee \neg(Xv \vee Ph^{\mathrm{in}}) \\
Mv^{\mathrm{out}} &= (Xv \vee Mh^{\mathrm{in}}) \wedge Ph^{\mathrm{in}} \\
&= Xv \wedge Ph^{\mathrm{in}}
\end{aligned}$$

  Viimeisessä askeleessa on hyödynnetty tietoa, että Mh^{in} ja Ph^{in} eivät voi olla 1 samanaikaisesti.

□

Lemman perusteella koko matriisin bittiesitys voidaan laskea seuraavalla algoritmilla:

for $i := 1$ **to** m **do**
  $Pv_{i0} := 1$; $Mv_{i0} := 0$
for $j := 1$ **to** n **do**
  $Ph_{0j} := 0$; $Mh_{0j} := 0$
  for $i := 1$ **to** m **do**
    $Xh_{ij} := Eq_{ij} \vee Mh_{i-1,j}$
    $Ph_{ij} := Mv_{i,j-1} \vee \neg(Xh_{ij} \vee Pv_{i,j-1})$
    $Mh_{ij} := Pv_{i,j-1} \wedge Xh_{ij}$
  for $i := 1$ **to** m **do**
    $Xv_{ij} := Eq_{ij} \vee Mv_{i,j-1}$
    $Pv_{ij} := Mh_{i-1,j} \vee \neg(Xv_{ij} \vee Ph_{i-1,j})$
    $Mv_{ij} := Ph_{i-1,j} \wedge Xv_{ij}$

Tämä algoritmi ei ole vielä aiempaa tehokkaampi, mutta sen sisäsilmukat voidaan bittirinnakkaistaa.

- Talletetaan sarakkeet Pv_{*j}, Mv_{*j}, Ph_{*j}, Mh_{*j} ja Eq_{*j} kukin yhteen bittivektoriin. Oletetaan, että $m \leq w$ ja vektoreissa on bitit positioissa $1, \ldots, m$.[1]

- Jälkimmäinen sisäsilmukka voidaan nyt suorittaa seuraavasti:

[1] Algoritmissa viitataan Ph- ja Mh-vektoreiden 0-bittiin, mutta niiden arvo on aina 0, mikä tulee siirto-operaatiossa oikein huomioonotetuksi.

$$\begin{aligned}
&Xv_{*j} := Eq_{*j} \vee Mv_{*,j-1}\\
&Pv_{*j} := (Mh_{*,j} << 1) \vee \neg(Xv_{*j} \vee \ (Ph_{*j} << 1))\\
&Mv_{*j} := (Ph_{*j} << 1) \wedge Xv_{*j}
\end{aligned}$$

Tässä on käytetty siirto-operaatiota $B << 1$, joka siirtää bittivektorissa B bitin kohdasta $i-1$ kohtaan i kaikilla $1 < i \leq |B|$ (ja asettaa $B_1 = 0$).

- Ensimmäisen silmukan bittirinnakkaistus aiheuttaa kuitenkin ongelman:

$$\begin{aligned}
&Xh_{*j} := Eq_{*j} \vee (Mh_{*j} << 1)\\
&Ph_{*j} := Mv_{*,j-1} \vee \neg(Xh_{*j} \vee \ Pv_{*,j-1})\\
&Mh_{*j} := Pv_{*,j-1} \wedge Xh_{*j}
\end{aligned}$$

Nyt vektoria Mh_{*j} käytetään Xh_{*j}:n laskemiseen ennen kuin se on laskettu. Laskentajärjestyksen muutos ei auta, koska Xh_{*j}:tä käytetään puolestaan Mh_{*j}:n laskemiseen.

- Kehästä päästään ulos laskemalla Xh_{*j} ilman Mh_{*j}:tä tai muita vasta myöhemmin arvonsa saavia bittivektoreita. Seuraava lemma määrittelee Xh_{*j}:n vektoreiden Eq_{*j} ja $Pv_{*,j-1}$ avulla.

Lemma 2.4.5

$$Xh_{ij} = \exists \ell \in [1, i] : Eq_{\ell j} \wedge (\forall x \in [\ell, i-1] : Pv_{x,j-1}).$$

Todistus. Todistetaan väite induktiolla i:n suhteen.

- Lemman 2.4.4 mukaan

$$\begin{aligned}
Xh_{ij} &= Eq_{ij} \vee Mh_{i-1,j}\\
Mh_{ij} &= Pv_{i,j-1} \wedge Xh_{ij}
\end{aligned}$$

- Alustus $i = 1$: Oikea puoli kutistuu muotoon Eq_{1j}. Koska $Mh_{0j} = 0$ kaikilla j, pätee $Xh_{1j} = Eq_{1j} \vee Mh_{0j} = Eq_{1j}$.

- Induktioaskel: Oletetaan, että $Xh_{i-1,j}$ on väitteen mukainen.

$$\begin{aligned}
Xh_{ij} &= Eq_{ij} \vee Mh_{i-1,j}\\
&= Eq_{ij} \vee (Pv_{i-1,j-1} \wedge Xh_{i-1,j})
\end{aligned}$$

Toisaalta

$$\begin{aligned}
&\exists \ell \in [1,i] : Eq_{\ell j} \wedge (\forall x \in [\ell, i-1] : Pv_{x,j-1}) \\
&= Eq_{ij} \vee \exists \ell \in [1,i-1] : Eq_{\ell j} \wedge (\forall x \in [\ell, i-1] : Pv_{x,j-1}) \\
&= Eq_{ij} \vee (Pv_{i-1,j-1} \wedge \exists \ell \in [1,i-1] : Eq_{\ell j} \wedge (\forall x \in [\ell, i-2] : Pv_{x,j-1}))
\end{aligned}$$

Väite seuraa nyt induktio-oletuksesta.

□

Lemman 2.4.5 kaava näyttää mahdottomalta laskea vakioajassa edes yhdelle bitille, saati sitten bittirinnakkaisesti koko vektorille Xh_{*j}. Se on kuitenkin mahdollista, mutta bittioperaatioita tarvitaan lisää:

- Merkitään xor-operaatiota symbolilla $\veebar$: $0 \veebar 1 = 1 \veebar 0 = 1$ ja $0 \veebar 0 = 1 \veebar 1 = 0$.
- Tulkitaan bittivektorit kokonaislukujen bittiesitykseksi siten, että vähiten merkitsevä bitti on positiossa 1. Bitit listataan eniten merkitsevästä alkaen. Esimerkiksi $B_{1\ldots m} = B_m B_{m-1} \cdots B_1$.
- Käytetään yhteenlaskua bittioperaationa. Esim. $0001 + 0111 = 1000$.

Lemma 2.4.6 *Merkitään $X = Xh_{*j}$, $E = Eq_{*j}$, $P = Pv_{*,j-1}$ ja olkoon $Y = (((E \wedge P) + P) \veebar P) \vee E$. Tällöin $X = Y$.*

Todistus. Kirjoitetaan X_i:n määritelmä ensin toiseen muotoon:

- $X_i = 1$, jos ja vain jos
  - a) $E_i = 1$ tai
  - b) $\exists \ell \in [1,i] : E_{\ell\ldots i} = 00\cdots01 \wedge P_{\ell\ldots i-1} = 11\cdots1$.
- $X_i = 0$, jos ja vain jos
  - c) $E_{1\ldots i} = 00\cdots0$ tai
  - d) $\exists \ell \in [1,i] : E_{\ell\ldots i} = 00\cdots01 \wedge P_{\ell\ldots i-1} \neq 11\cdots1$.

Osoitetaan tapauskohtaisesti, että $Y_i = X_i$:

a) Y:n määritelmän loppu "$\vee E$" takaa, että $Y_i = 1$.

b) Olkoon $\mathtt{b}$ bitti P_i eli $P_{\ell\ldots i} = \mathtt{b11...1}$ ja $\mathtt{c}$ bittien $1\ldots,\ell-1$ yhteenlaskusta mahdollisesti tuleva muistibitti.

$$
\begin{array}{rcl}
 & & \quad i \qquad \ell \\
E_{\ell\ldots i} & = & \mathtt{00...01} \\
P_{\ell\ldots i} & = & \mathtt{b1...11} \\
(E \wedge P)_{\ell\ldots i} & = & \mathtt{00...01} \\
((E \wedge P) + P)_{\ell\ldots i} & = & \mathtt{\bar{b}0...0c} \\
(((E \wedge P) + P) \veebar P)_{\ell\ldots i} & = & \mathtt{11...1\bar{c}} \\
Y = ((((E \wedge P) + P) \veebar P) \vee E)_{\ell\ldots i} & = & \mathtt{11...11}
\end{array}
$$

missä $\bar{\mathtt{b}}$ on $\mathtt{b}$:n negaatio.

c) Koska kaikilla bittivektoreilla B, $0 \wedge B = 0$ ja $0 + B = B$, saadaan $Y = (((0 \wedge P) + P) \veebar P) \vee 0 = (P \veebar P) \vee 0 = 0$.

d) Tarkastellaan kohdan b) laskentaa. Sen ideana on, että yhteenlaskussa positiossa ℓ syntyvä muistibitti kantautuu aina positioon i asti ja tuottaa sinne bitin $\bar{\mathtt{b}}$. Erona on nyt, että ainakin yksi bitti P_k, $\ell \le k < i$, on 0. Jos $k = \ell$, niin $P_\ell = (E \wedge P)_\ell = 0$ ja muistibittiä ei synny (riippumatta bitistä c). Jos taas $k > \ell$, muistibitti pysähtyy positioon k. Kummassakin tapauksessa $((E \wedge P) + P)_i = \mathtt{b}$ ja siten $Y_i = 0$.

$\square$

- Määritellään $Peq[a]$, $a \in \Sigma$, on bittivektori, jossa on 1 positiossa i jos ja vain jos $p_i = a$. Tällöin $Eq_{*j} = Peq[s_j]$.

Algoritmi 2.4.7 *Myers*

Syöte: P, S, k, $|P| \le w$, missä w on tietokoneen sanan pituus
Tuloste: P:n likimääräiset esiintymäkohdat k virheellä S:ssä

Laske $Peq[a]$ jokaiselle $a \in \Sigma$
$Pv := 1^m$; $Mv := 0$; $Sc := m$
for $j := 1$ **to** n **do**
    $Eq := Peq[S_j]$
    $Xh := (((Eq \wedge Pv) + Pv) \veebar Pv) \vee Eq$;
    $Ph := Mv \vee \neg(Xh \vee Pv)$
    $Mh := Pv \wedge Xh$;
    $Xv := Eq \vee Mv$
    $Pv := (Mh << 1) \vee \neg(Xv \vee (Ph << 1))$
    $Mv := (Ph << 1) \wedge Xv$
    $Sc := Sc + Ph(m) - Mh(m)$
    if $Sc \le k$ **then** *write* j

Lause 2.4.8 *Olkoon hahmo $P = p_1 \cdots p_m$ ja teksti $S = s_1 \cdots s_n$. Likimääräinen hahmonsovitus k virheellä voidaan ratkaista Algoritmilla 2.4.7 ajassa $\mathcal{O}(\lceil m/w \rceil n)$, missä w on tietokoneen sananpituus bitteinä.*

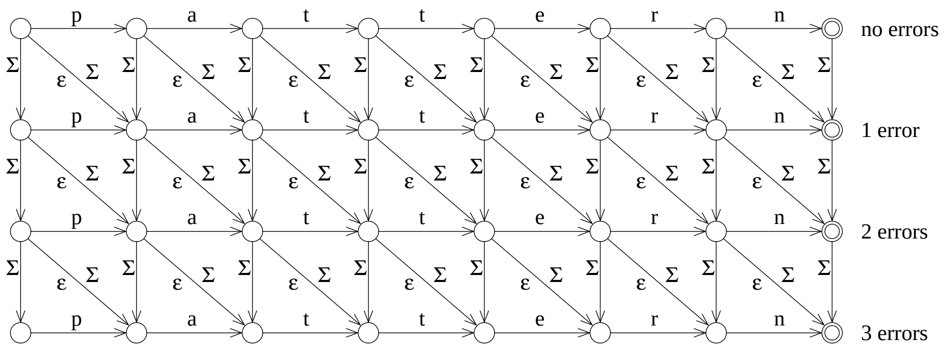
Todistus. On selvää, että Algoritmi 2.4.7 toimii ajassa $\mathcal{O}(n)$ kun $m \leq w$. Algoritmia voidaan helposti simuloida pidemmillä hahmoilla käyttäen useaa tietokonesanaa vektorien Pv, Eq, jne., esittämiseen. □

Huomautus 2.4.9 *Rajoittamalla laskenta kullakin sarakkeella viimeiseen arvoon k asti (kuten Algoritmissa 2.3.6), voidaan Algoritmia 2.4.7 tehostaa toimimaan odotusarvoisesti $\mathcal{O}(\lceil k/w \rceil n)$ ajassa.*

Huomautus 2.4.10 *Monessa laskennan mallissa oletetaan, että $w = \Omega(\log n)$ (jotta muistiosoite mahtuu vakiomäärään sanoja). Tällöin siis Algoritmin 2.4.7 aikavaatimus on oikeastaan $\mathcal{O}(mn/\log n)$. Muitakin samanlaisia algoritmeja on. Masekin ja Patersonin (J. Comput. Syst. Sci. 20:18–31, 1980) "neljän venäläisen" menetelmään perustuvalla algoritmilla aikavaatimus on jopa $\mathcal{O}(mn/\log_\sigma^2 n)$.*

- On olemassa myös muita bittirinnakkaisuuteen perustuvia algoritmeja likimääräiseen hahmonsovitukseen.

- Kaksi eri algoritmia perustuu seuraavan epädeterministisen automaatin simulointiin:



- Wun ja Manberin (CACM 35(10):83–91, 1992) algoritmi, joka koodaa kunkin automaatin rivin erillisellä bittivektorilla, voidaan nähdä shift-or-algoritmin yleistyksenä. Sen aikavaatimus on $\mathcal{O}(k\lceil m/w \rceil n)$.

- Baeza-Yatesin ja Navarron (Algorithmica 23(2):127–158, 1999) algoritmi koodaa automaatin lävistäjät bittivektoreilla, jotka voi pakata yhdeksi pitkäksi vektoriksi. Aikavaatimus on $\mathcal{O}(\lceil km/w \rceil n)$.