1.8 Hahmonsovitus jokerimerkeillä

M. Fischer & M. Paterson. String matching and other products. Teoksessa *R. Karp, editor, SIAM AMS Complexity of Computation*, sivut 113–125, 1974.

Edellä käsiteltiin tapausta, jossa hahmojen joukko esitettiin säännöllisenä lausekkeena. Luonnollinen kysymys on

- löytyykö sellaisia ei-triviaaleja säännöllisten lausekkeiden aliluokkia, jolla hahmontunnistus voitaisiin toteuttaa oleellisesti nopeammin kuin Algoritmilla 1.7.5?

Tutkitaan seuraavaa säännöllisen lausekkeen erikoistapausta:

- Hahmo P voi sisältää aakkoston Σ merkkien lisäksi *jokerimerkkejä* $\# \notin \Sigma$, jotka täsmäävät minkä tahansa aakkoston $\Sigma \cup \{\#\}$ merkin kanssa.

Esimerkki 1.8.1 *Hahmolle $P =$ `#oke#i` löytyy 3 esiintymää tekstissä $S =$ `sokeripokeritokeni`.*

Ratkaisuja

- Triviaalialgoritmi. Aikavaatimus $\mathcal{O}(nm)$.
- Shift-or-algoritmi. Aikavaatimus $\mathcal{O}(n\lceil m/w \rceil)$.
- Fischer-Paterson-algoritmi. Aikavaatimus $\mathcal{O}(n \log^2 m \log\log m)$.

Lause 1.8.2 *Jos ainoa pääsy syötteeseen on merkkivertailu, tarvitaan $\Omega(nm)$ vertailua löytämään jokerimerkkejä sisältävän hahmon P, $|P| = m$ kaikki esiintymät jokerimerkkejä sisältävästä tekstistä S, $|S| = n$.*

Todistus. Olkoon hahmo $P = \#^m$ ja teksti $S = \#^n$ (siis pelkkiä jokerimerkkejä). Hahmon esiintymät alkavat kaikista positioista $0, \ldots, n-m$. Jos jossain algoritmissa vertailua "$p[i] = s[j]$", jollain $(i, j) \in [0 \ldots m) \times [0, \ldots n)$, ei ole suoritettu, niin voidaan korvata $p[i]$ ja $s[j]$ merkeillä a ja b, $a \neq b$; minkään muun vertailun tulos ei muutu, joten kyseinen algoritmi tuottaa virheellisen tuloksen (yksi esiintymä liikaa) muunnetulla syötteellä. Siis kaikki vertailut "$p[i] = s[j]$", $0 \leq i < m$, $i \leq j \leq n - m + i$, pitää suorittaa. $\square$

Fischer-Paterson-algoritmi

Lauseesta 1.8.2 seuraa, että ongelman ratkaisuun täytyy käyttää jotain muuta laskennan mallia kuin merkkivertailua. Fischer-Paterson-algoritmi palauttaa ongelman kokonaislukujen kertolaskuun, johon on tunnetusti kehitetty useita algoritmeja eri laskennan malleissa.

Aloitetaan merkkijonojen kertolaskusta:

- Määritellään merkkijonojen U ja V *kertolasku* $Z = U \circ V$ siten, että

$$Z[k] = AND_{i+j=k}(U[i] = V[j]),$$

  missä yhtäsuuruusvertailu tuottaa tuloksen 1 (tosi) jos ja vain jos $U[i]$ ja $V[j]$ ovat samoja merkkejä, tai jompikumpi tai molemmat ovat jokerimerkkejä, muutoin vertailun tulos on 0 (epätosi).

- Tämä tarkoittaa, että $Z[k] = 1$ jos ja vain jos $U[i] = V[j]$ **kaikilla** $i + j = k$, muuten $z[k] = 0$.

- Olkoon $P' = p'_0p'_1 \cdots p'_{m-1}$ hahmon $P = p_0p_1 \cdots p_{m-1}$ *käänteisjono*, eli $p'_0p'_1 \cdots p'_{m-1} = p_{m-1}p_{m-2} \cdots p_0$.

- Tutkitaan P':n ja tekstin S kertolaskua $Z = P' \circ S$.

- Huomataan, että $Z[k] = 1$ on yhtäpitävää sen kanssa, että hahmon P esiintymä loppuu tekstin kohtaan k, kun $m - 1 \leq k < n$.

- Tästä seuraa, että hahmonsovitus (jokerimerkkien kanssa tai ilman) voidaan palauttaa $\circ$-operaation laskentaan.

Tarkastellaan nyt miten $\circ$-operaatio voidaan laskea tehokkaasti.

- Määritellään merkkijonolle U ja merkille a bittivektori $occur(a, U)$ siten, että vektorin i:nnes arvo on 1 jos ja vain jos $U[i] \equiv a$ (Merkintä $\equiv$ tarkoittaa että merkin $U[i]$ pitää olla *täsmälleen* a; jos $U[i] \equiv \#$ niin vektorin i:nnes arvo on aina 0).

- Olkoon X ja Y kaksi *bittivektoria*, jotka sisältävät siis vain arvoja 0 ja 1. Määritellään $Z = X \diamond Y$ siten, että

$$Z[k] \quad = \quad OR_{i+j=k}(X[i] = 1 = Y[j]).$$

- Tämä tarkoittaa, että $Z[k] = 1$ jos ja vain jos $X[i] = 1 = Y[j]$ **jollain** $i + j = k$, muuten $Z[k] = 0$.

- Merkitään

  $$OCCUR_{a,b}(X,Y) = occur(a,U) \diamond occur(b,V).$$

  Tämä vektori kertoo ne esiintymäkohdat, joissa U:n merkki a ja V:n merkki b osuvat kohdakkain.

- Kahden merkkijonon U ja V kertolasku $U \circ V$ voidaan nyt esittää $\diamond$-operaatioiden avulla:

  $$U \circ V = \neg OR_{a,b \in \Sigma, a \neq b} OCCUR_{a,b}(U,V). \tag{1.1}$$

- Jos oletetaan, että aakkoston koko on vakio, voidaan $\circ$-operaatio laskea vakiomäärällä ($=\mathcal{O}(\sigma^2) = \mathcal{O}(1)$) $\diamond$-operaatioita.

Vielä pitää osoittaa miten $\diamond$-operaatiot voidaan toteuttaa tehokkaasti. Tämä tehdään palauttamalla laskenta tavalliseen kokonaislukujen kertolaskuun $*$:

- Olkoon X ja Y kaksi bittivektoria, $|X| = m \leq |Y| = n$.
- Olkoon $\ell = \lceil \log_2(m+1) \rceil$.
- Lisätään $\ell - 1$ 0:llaa kunkin peräkkäisen bitin väliin X:ssä ja Y:ssä.
- Merkitään x':lla ja y':lla kokonaislukuja, joiden binääriesitys on muodostettu edellä X:stä ja Y:stä.
- Olkoon z' kokonaisluku $z' = x' * y'$.
- Bittivektori $Z = X \diamond Y$ voidaan nyt lukea z':n binääriesityksestä: Luetaan ℓ ensimmäistä bittiä. Tämä luku kokonaislukuna on $match[0] = \sum_{i+j=0}(X[i] = 1 = Y[j])$. Seuraavat ℓ bittiä kokonaislukuna antaa luvun $match[1] = \sum_{i+j=1}(X[i] = 1 = Y[j])$, jne. Siis $Z[k] = 1$ jos ja vain jos $match[k] > 0$, muuten $Z[k] = 0$.
- Edellä ℓ:n arvo valittiin siten, että peräkkäisten kenttien arvot eivät pääse sekoittumaan.

Lause 1.8.3 *Olkoon jokerimerkkejä sisältävien hahmon ja tekstin pituudet m ja n, $m \leq n$. Tällöin hahmon esiintymät tekstisssä voidaan löytää ajassa $IM(n \log m, m \log m)$, missä $IM(N,M)$ on kahden N ja M, $N \geq M$, bittiä pitkien kokonaislukujen kertolaskuun tarvittava aika.*

Schönhage-Strassen kertolaskualgoritmi käyttää $O(N \log M \log\log M)$ bittioperaatiota, mistä seuraa $O(n \log^2 m \log\log m)$ algoritmi hahmonsovitukselle jokerimerkkien kanssa. Kertolaskussa voidaan käyttää myös tietokoneen omaa kertolasku-operaatiota, mutta tällöin aikavaatimukseksi tulee $O(mn)$, eli sama kuin triviaalialgoritmilla.

1.9 Kaksiulotteinen hahmonsovitus

R. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6:168–170, 1977.

T. Baker. A technique for extending rapid exact-match string matching to arrays more than one dimension. *SIAM Journal on Computing*, 7:533–541, 1978.

Tehtävänä on etsiä $m \times m$ hahmon P esiintymät $n \times n$ tekstistä S:

$$P = \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{array} \qquad S = \begin{array}{ccccccccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}$$

Bird-Baker-algoritmi

1. Selataan S:n rivit AC-automaatilla käyttäen hahmoina P:n rivejä; $n \times (n - m + 1)$ aputaulukkoon merkitään tunnistetut rivit.

2 Selataan aputaulukko KMP-algoritmilla sarakkeittain.

aputaulukko:
0: ei hahmon rivi
$i > 0$: hahmon i's rivi

```
3 0 0 3 1 0
0 0 0 0 0 2
4 0 3 0 0 0
0 2 0 3 0 0
0 3 1 2 0 0
0 0 2 0 0 0
0 0 3 0 0 0
0 0 4 0 3 1
3 0 0 0 0 1
```

etsitään
aputaulukosta:

```
1
2
3
4
```

- Aikavaatimus: $\mathcal{O}(n^2 + m^2)$.

Monia muitakin algoritmeja:

- aakkostoriippumattomia
- keskimäärin nopeampia

Luku 2

Likimääräinen hahmonsovitus

Useissa sovelluksissa ei ole mielekästä etsiä vain hahmon P tarkkoja esiintymiä, vaan pyritään löytämään kaikki tekstin osajonot, jotka ovat riittävän lähellä hahmoa P. Jonojen samankaltaisuuden eräs mitta on *editointietäisyys*.

2.1 Editointietäisyys

Olkoon $A = a_1 \cdots a_m \in \Sigma^*$ “lyöntivirheitä sisältävä jono” ja $B = b_1 \cdots b_n \in \Sigma^*$ “virheetön jono”.

- Merkkijonojen A ja B välinen editointietäisyys on pienin editointioperaatioiden lukumäärä, joka tarvitaan korjaamaan virheellinen jono A virheettömään muotoon B.

- Editointioperaatiot

  P. *Poisto*: A:n merkkiä a_i ei vastaa mikään B:n merkki, $a_i \to \epsilon$.

  L. *Lisäys*: B:n merkkiä b_j ei vastaa mikään A:n merkki, $\epsilon \to b_j$.

  M. *Muutos*: A:n merkkiä a_i vastaa B:n merkki b_j, $a_i \neq b_j$, $a_i \to b_j$.

- Operaatiot M eivät saa mennä ristiin:

  – Jos $a_i \to b_j$ ja $a_{i'} \to b_{j'}$, niin $i < i'$, jos ja vain jos $j < j'$.

- Usein on myös hyödyllistä määritellä “operaatio”:

  T. *Täsmäys*: A:n merkkiä a_i vastaa B:n merkki b_j, $a_i = b_j$, $a_i \to b_j$.

Määritelmä 2.1.1 *Merkkijonojen A ja B välinen editointietäisyys $D(A, B)$ on pienin editointioperaatioiden P, L ja M määrä, joka tarvitaan muuntamaan A B:ksi.*

- Editointietäisyydestä käytetään myös nimitystä *Levenshtein*-etäisyys.
- Jos sallitaan pelkkä operaatio M (muutos), etäisyydestä käytetään nimitystä *Hamming*-etäisyys.
- Voidaan osoittaa, että D on *metriikka*:
  1) $D(A, B) \geq 0$,
  2) $D(A, B) = 0$, jos ja vain jos $A = B$,
  3) $D(A, B) = D(B, A)$,
  4) $D(A, C) \leq D(A, B) + D(B, C)$.

Esimerkki 2.1.2 $A = \texttt{abba}$, $B = \texttt{bbb}$

```
abba                        abba
|:||   3 operaatiota        |::|   2 operaatiota
bb b                        bbb
```

Tarvitaan vähintään yksi poisto, koska $|A| = |B| + 1$, ja vähintään yksi lisäys tai muutos, koska B:ssä on yksi b enemmän kuin A:ssa. Tämän ja ylläolevan perusteella $D(A, B) = 2$.

Editoinnin havainnollistaminen

1. Jäljitys (trace)

```
i n d u s t r y

i n t e r e s t
```

- Viivat eivät saa mennä ristiin. (Miksi?)

2. Kohdistus

```
indust-ry--
in---terest
```

3. Operaatiojono

```
TTPPPTLTMPP
indust-ry--
in---terest
```

4. Operaatioiden luettelo

industry	$d \to \epsilon$	intery	$\epsilon \to e$
inustry	$u \to \epsilon$	interey	$y \to s$
instry	$s \to \epsilon$	interes	$\epsilon \to t$
intry	$\epsilon \to e$	interest	

2.2 Editointietäisyyden laskeminen

Merkitään $d_{ij} = D(a_1 \cdots a_i, b_1 \cdots b_j)$, $0 \leq i \leq m$, $0 \leq j \leq n$.

Lause 2.2.1 *Editointietäisyydet d_{ij} ja erityisesti $d_{mn} = D(A, B)$ voidaan laskea palautuskaavasta*

$$d_{ij} = \min\{d_{i-1,j-1} + (\textbf{\textit{if}}\ a_i = b_j\ \textbf{\textit{then}}\ 0\ \textbf{\textit{else}}\ 1), d_{i-1,j} + 1, d_{i,j-1} + 1\}, \quad (2.1)$$

missä $1 \leq i \leq m$, $1 \leq j \leq n$, kun alustetaan

$$\begin{aligned} d_{00} &= 0, \\ d_{i0} &= i,\ 1 \leq i \leq m,\ ja \\ d_{0j} &= j,\ 1 \leq j \leq n. \end{aligned}$$

Todistus. Induktio.
1. Alustus on oikein:

$$\begin{aligned} d_{00} &= D(\epsilon, \epsilon) = 0, \\ d_{i0} &= D(a_1 a_2 \cdots a_i, \epsilon) = |(a_1 \to \epsilon)(a_2 \to \epsilon) \cdots (a_i \to \epsilon)| = i, \\ d_{0j} &= D(\epsilon, b_1 b_2 \cdots b_j) = |(\epsilon \to b_1)(\epsilon \to b_2) \cdots (\epsilon \to b_j)| = j. \end{aligned}$$

2. Induktioaskel:

- Oletetaan, että $d_{i',j'}$ on oikein, kun $i' + j' < i + j$. Tarkastellaan $a_1 a_2 \cdots a_i$:n ja $b_1 b_2 \cdots b_j$:n viimeisten merkkien mahdollisia editointeja. Tapauksia on kolme:

(i) $a_i \rightarrow b_j$ (muutos tai täsmäys): Lyhimmän tällaisen editointijonon pituus on $d_{i-1,j-1} + (\textbf{if}\ \ a_i = b_j \textbf{ then } 0 \textbf{ else } 1)$.

(ii) $a_i \rightarrow \epsilon$ ja b_j:hin kohdistuu muutos tai täsmäys: Lyhimmän tällaisen editointijonon pituus on $d_{i-1,j} + 1$.

(iii) $\epsilon \rightarrow b_j$ ja a_i:hin kohdistuu muutos tai täsmäys: Lyhimmän tällaisen editointijonon pituus on $d_{i,j-1} + 1$.

- Tapausta, jossa samanaikaisesti $a_i \rightarrow \epsilon$ ja $\epsilon \rightarrow b_j$ ei tarvitse ottaa huomioon, koska sen pituus on $d_{i-1,j-1} + 2 > d_{i-1,j-1} + 1$.

- Lyhin editointijono $a_1 a_2 \cdots a_i \rightarrow b_1 b_2 \cdots b_j$ on pituudeltaan kohtien (i), (ii) ja (iii) minimi, joten kaava (2.1) pätee. □

Esimerkki 2.2.2 $A =$ *ballad*, $B =$ *handball*

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

$D(A, B) = d_{mn} = d_{6,8} = 6.$

Arvojen (d_{ij}), $0 \leq i \leq m$, $0 \leq j \leq n$, laskenta voidaan toteuttaa *taulukointina* eli käyttämällä ns. *dynaamista ohjelmointia*:

Algoritmi 2.2.3 *$D(A, B)$:n laskenta dynaamisella ohjelmoinnilla, perusversio.*

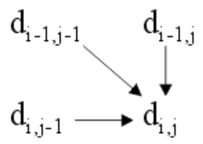
Syöte: $A = a_1 a_2 \cdots a_m$, $B = b_1 b_2 \cdots b_n$
Tuloste: Matriisi (d_{ij}), $0 \leq i \leq m$, $0 \leq j \leq n$, $d_{mn} = D(A, B)$
for $i := 0$ **to** m **do** $d_{i0} := i$; (* alustukset *)
for $j := 1$ **to** n **do** $d_{0j} := j$;
for $j := 1$ **to** n **do**
    for $i := 1$ **to** m **do**
        $d_{ij} := \min\{d_{i-1,j-1} + (\textbf{if } a_i = b_j \textbf{ then } 0 \textbf{ else } 1), d_{i-1,j} + 1, d_{i,j-1} + 1\}$

- Aikavaatimus on $\Theta(mn)$.

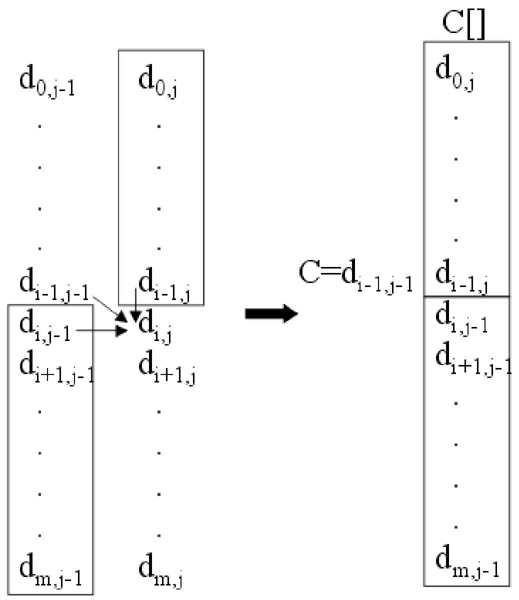
- Suoraviivaisen ratkaisun tilavaatimus on $\Theta(mn)$ (eli matriisin (d_{ij}) vaatima tila).

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.
(1) **for** $i := 0$ **to** m **do** $C[i] := i$; (* alustus $j = 0$*)
(2) **for** $j := 1$ **to** n **do**
(3) $C := C[0]; C[0] := j$; (* alustus $(i = 0)$ *)
(4) **for** $i := 1$ to m **do**
(* invariantti: $C = d_{i-1,j-1}$, $C[i-1] = d_{i-1,j}$, $C[i] = d_{i,j-1}$ *)
(5) $tmp := \min(C + (\textbf{if } a_i = b_j \textbf{ then } 0 \text{ else } 1), C[i-1] + 1, C[i] + 1)$;
(6) $C := C[i]$; (* $d_{i,j-1}$ talteen *)
(7) $C[i] := tmp$;
(8) *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: $a_i \rightarrow b_j$,

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

(3) lisäys: $\epsilon \rightarrow b_j$.

- 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 Lävistäjämenetelmä

E. Ukkonen. Algorithms for approximate string matching. *Information and Control* 64(1–3):100–118, 1985.

- Matriisin (d_{ij}) rakenteessa on säännönmukaisuus, joka johtaa perusalgoritmeja tehokkaampaan ratkaisuun erityisesti siinä mielenkiintoisessa tapauksessa, että $D(A, B)$ on pieni verrattuna arvoihin m ja n.

- Matriisin (d_{ij}) *lävistäjään* k, $-m \le k \le n$, kuuluvat ne alkiot d_{ij} joilla $j - i = k$.

d		c	b	a	c	a	b	a
	0	1	2	3	4	5	6	7
b	1	1	1	2	3	4	5	6
a	2	2	2	1	2	3	4	5
a	3	3	3	2	2	2	3	4
c	4	3	4	3	2	3	3	4

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

Lemma 2.3.1 *Jokaisella d_{ij}, $1 \le i \le m$, $1 \le j \le n$: $d_{ij} = d_{i-1,j-1}$ tai $d_{ij} = d_{i-1,j-1} + 1$. (Huomaa, että d_{ij} ja $d_{i-1,j-1}$ ovat samalla lävistäjällä.)*

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

$$d_{i-1,j-1} \quad \le \quad d_{ij}.$$

Tämä todistetaan induktiolla $i + j$:n suhteen:

a) Minimointi antaa d_{ij}:n $d_{i-1,j-1}$:stä, joten $d_{ij} \ge d_{i-1,j-1}$.

b) Minimointi antaa d_{ij}:n $d_{i-1,j}$:stä. Tämä voi tapahtua vain, jos $i > 1$, koska $d_{1j} \le d_{0,j-1} + 1 < d_{0j} + 1$. Tällöin pätee

$$d_{ij} = d_{i-1,j} + 1 \ge d_{i-2,j-1} + 1 \ge d_{i-1,j-1}$$

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

c) Minimointi antaa d_{ij}:n $d_{i,j-1}$:stä. Kuten b-kohta.

□

- Lemma 2.3.1 johtaa tehokkaaseen (d_{ij}):n esitykseen; talletetaan vain ne lävistäjien kohdat, joissa arvo kasvaa.

- Matriisin laskenta voidaan samalla rajoittaa niihin d_{ij}, joilla $d_{ij} \leq d_{mn}$, sillä vain näillä voi olla vaikutusta arvoon d_{mn}, koska vain tällainen d_{ij} voi olla minimoivalla reitillä.

- Käytetään matriisia (f_{kp}), joka tallettaa (d_{ij}):n lävistäjittäin:

  – f_{kp} on suurin i siten, että matriisin (d_{ij}) lävistäjällä k arvo p ulottuu riville i, eli

$$f_{kp} = \max\{i \mid d_{ij} = p \text{ ja } k = j - i\}. \tag{2.2}$$

  – Alustus: $f_{0,-1} = -1$ ja $f_{k,p} = -\infty$, kun $p \leq |k| - 1$.

  – Huomaa: $d_{mn} = p$, jolla $f_{n-m,p} = m$.

Esimerkki 2.3.2 *Seuraavassa on esitettynä matriisi (d_{ij}) ja sitä vastaava (f_{kp}) ('-' tarkoittaa arvoa $-\infty$).*

d		c	b	a	c	a	b	a
	0	1	2	3	4	5	6	7
b	1	1	1	2	3	4	5	6
a	2	2	2	1	2	3	4	5
a	3	3	3	2	2	2	3	4
c	4	3	4	3	2	3	3	4

f	-1	0	1	2	3	4	5	6	7
-4	-	-	-	-	-	4			
-3	-	-	-	-	4				
-2	-	-	-	2	3	4			
-1	-	-	1	2	4				
0	-1	0	1	4					
1	-	-	2	3	4				
2	-	-	-	3	4				
3	-	-	-	-	3	4			
4	-	-	-	-	-	3			
5	-	-	-	-	-	-	2		
6	-	-	-	-	-	-	-	1	
7	-	-	-	-	-	-	-	-	0

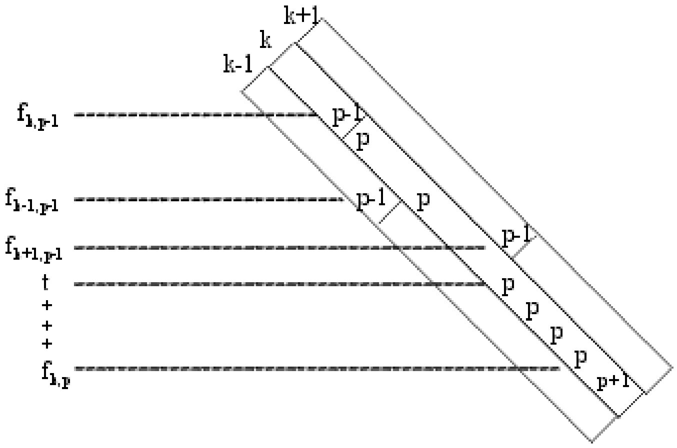
Matriisin (f_{kp}) laskenta sarakkeittain

- Olkoon (f_{kp}):n $p-1$:s sarake $f_{*,p-1}$ laskettu. Halutaan laskea f_{kp}.
  - Laskenta etenee matriisissa (d_{ij}) vyöhykkeittäin: ensin käsitellään kaikki d_{ij}:t, jotka ovat nollia, sitten kaikki d_{ij}:t, jotka ovat ykkösiä jne., kunnes saavutetaan d_{mn}.

- Lävistäjällä k arvot p ulottuvat ainakin riville t, missä

$$t = \max\{f_{k,p-1}+1, f_{k-1,p-1}, f_{k+1,p-1}+1\}, \tag{2.3}$$

  mikäli lävistäjä k ulottuu näin pitkälle. Ylläoleva kaava seuraa, kun tarkastelee arvojen $f_{k,p-1}$, $f_{k-1,p-1}$ ja $f_{k+1,p-1}$ kaikkia mahdollisia järjestyksiä. Alla olevassa kuvassa on havainnollistettu yhtä mahdollista tilannetta.



- Jos lisäksi rivillä $i = t+1$ on $a_i = b_j$ lävistäjällä $k = j - i$, ei d_{ij} voi kasvaa, joten arvo p ulottuu ainakin riville $t+1$. Tätä voidaan toistaa, niin kauan kuin $a_i = b_j$ lävistäjällä $k = j - i$. Siis

$$f_{kp} = \max\{i \mid i \leq m \text{ ja } i+k \leq n \text{ ja } A[t+1 \ldots i] = B[t+k+1 \ldots i+k]\}. \tag{2.4}$$

- Saadaan seuraava menetelmä f_{kp}:n laskemiseksi (Algoritmi A):

(1) $t := \max\{f_{k,p-1}+1, f_{k-1,p-1}, f_{k+1,p-1}+1\}$;
(2) **while** $t < m$ **and** $t + k < n$ **and** $a_{t+1} = b_{t+1+k}$ **do** $t := t+1$;
(3) $f_{kp} := t$;

- Nyt (f_{kp}) voidaan laskea sarakkeittain seuraavalla algoritmilla:

Algoritmi 2.3.3 *Editointietäisyys lävistäjämenetelmällä*

Syöte: Jonot $A = a_1 \cdots a_m$, $B = b_1 \cdots b_n$, $m \leq n$
Tuloste: $p = D(A, B)$ ja matriisi (f_{kp}) siltä osin, kuin sitä on tarvittu p:n laskemiseksi.
Oletus: Alustamaton arvo $f_{kp} = -\infty$.
(1) $p := -1$; $f_{0p} := -1$;
(2) **while** $f_{n-m,p} < m$ **do**
(3) $\quad p := p + 1$;
(4) $\quad$ **for** $k := -p$ **to** p **do** laske f_{kp} Algoritmilla A;

- Algoritmissa 2.3.3 on käytetty huomiota, että p voi esiintyä vain lävistäjillä k, $-p \leq k \leq p$.

- *Aikavaatimus:* Algoritmia A kutsutaa $\mathcal{O}(d^2)$ kertaa, missä $d = D(A, B)$. Kukin kutsu vaati vakioajan lukuunottamatta rivin (2) silmukkaa. Silmukassa selataan lävistäjää k eteenpäin ja samaa kohtaa ei koskaan selata kahdesti. Silmukkaa toistetaan siten kutakin lävistäjää kohden $\mathcal{O}(m)$ kertaa ja yhteensä enintään $\mathcal{O}(dm)$ kertaa. Siis aikavaatimus on $\mathcal{O}(d(d + m))$

Algoritmia A kutsutaan silloinkin, kun lävistäjä on jo saatu valmiiksi. Välttämällä sitä, aikavaatimusta voidaan parantaa tapauksessa $m < d$.

- Lävistäjä k on valmis, kun $f_{kp} = m$ (pohjarivi saavutettu) tai $f_{kp} + k = n$ (oikea laita saavutettu). Tarkastellaan tapausta $f_{kp} = m$:
  - Lävistäjää k ei tarvitse enää käsitellä.
  - Myöskään lävistäjiä $h < k$ ei tarvitse enää käsitellä, vaikka ne eivät olisikaan valmiit, koska ne eivät enää voi vaikuttaa lävistäjään $n - m$.

- Tapaus $f_{kp} + k = n$ on vastaava.

- Saadaan seuraava algoritmi:

Algoritmi 2.3.4 *Editointietäisyys lävistäjämenetelmällä (parannettu)*
Syöte: Jonot $A = a_1 \cdots a_m$, $B = b_1 \cdots b_n$, $m \leq n$
Tuloste: $p = D(A, B)$ ja matriisi (f_{kp}) siltä osin, kuin sitä on tarvittu p:n laskemiseksi.
Oletus: Alustamaton arvo $f_{kp} = -\infty$.
(1) $p := -1$; $f_{0p} = -1$; $low := -\infty$; $high := \infty$;
(2) **while** $f_{n-m,p} < m$ **do**
(3) $\quad p := p + 1$;
(4) $\quad$ **for** $k := \max(-p, low)$ **to** $\min(p, high)$ **do**
(5) $\quad\quad$ laske f_{kp} Algoritmilla A;
(6) $\quad\quad$ **if** $f_{kp} = m$ **then** $low := k + 1$;
(7) $\quad\quad$ **if** $f_{kp} + k = n$ **then** $high := k - 1$;

- Huomaa, että nyt $f_{kp} \leq m$ ja $f_{kp} + k \leq n$ kaikilla k ja p. (Miksi?)

- *Parannetun algoritmin aikavaatimus:* Jokainen Algoritmin A kutsu etenee lävistäjällä eli $f_{kp} > f_{k,p-1}$. Koska $f_{kp} \leq m$, kutsuja on yhteensä enintään $m+1$ lävistäjää kohden. Näin saadaan kokonaisajaksi $\mathcal{O}(dm)$.

- Tilavaatimus on $\mathcal{O}(d \min(d, m))$ edellyttäen, että kullekin (f_{kp}):n sarakkeelle varataan tila vasta kun sen koko tiedetään.

- Tilavaatimusta voidaan vielä pienentää, kun huomataan, että arvon f_{kp} laskemiseen tarvitaan vain edellisen sarakkeen arvoja $f_{k,p-1}$, $f_{k-1,p-1}$ ja $f_{k+1,p-1}$. Voidaan siis tallettaa vain edellinen ja nykyinen sarake. Näiden yhteiskoko on $\mathcal{O}(d)$.

- *Huom.* Alkuperäisartikkelissa on esitetty hieman erilainen tapa tehostaa Algoritmin 2.3.3 toimintaa. Syksyn 2003 kurssin muistiinpanoissa on kolmas tapa. Tuloksena kaikissa on kuitenkin sama aika- ja tilavaatimus.

Lause 2.3.5 *Olkoon $|A| = m \leq n = |B|$. Lävistäjämenetelmällä (Algoritmi 2.3.4) voidaan etäisyys $d = D(A, B)$ laskea ajassa $\mathcal{O}(dm)$ ja tilassa $\mathcal{O}(d)$.*

Editointipolun tulostaminen

- Algoritmin A rivin 1 kolme tapausta vastaavat kolmea editointioperaatiota muutos, lisäys ja poisto.

- Minimiarvoa $d = D(A, B)$ vastaavat editointioperaatiot voidaan tulostaa, jos kuhunkin arvoon f_{kp} liitetään linkki niihin arvoista $f_{k,p-1}$, $f_{k-1,p-1}$ ja $f_{k+1,p-1}$, joista saatiin maksimi.

- Tällöin tilavaatimuksi tulee $\mathcal{O}(d \min(d, m))$.