58093-3 Merkkijonomenetelmät (4 ov), kevät 2005

Juha Kärkkäinen[1]

2. helmikuuta 2005

[1]Perustuu Jorma Tarhion, Esko Ukkosen, Tibor Hegedüksen ja Veli Mäkisen aiempiin muistiinpanoihin.

Yleistä kurssista

- Kurssi on valinnainen laudatur-kurssi, joka soveltuu erityisesti algoritmien erikoistumislinjan sekä bioinformatiikan ja laskennallisen biologian suuntautumisvaihtoehdon opiskelijoille.
- Esitiedoiksi oletetaan kurssien Tietorakenteet sekä Ohjelmoinnin ja laskennan perusmallit tiedot. Lisäksi hyödyllisiä ovat kurssit Laskennan teoria sekä Algoritmien suunnittelu ja analyysi.
- Luennot: Juha Kärkkäinen | 31.01.-13.04. | ma, ke 12-14 | C222
- Harjoitukset:
  - Pasi Rastas | 07.02.-18.04. | ma 10-12 | C221
  - Pasi Rastas | 08.02.-19.04. | ti 10-12 | DK117
- Kurssiin kuuluu ohjelmointiprojekti, jonka tulokset raportoidaan postereina. Tarkemmat tiedot ja aiheet tulevat myöhemmin kurssin kotisivulle `http://www.cs.helsinki.fi/u/tpkarkka/mjm/k05/`.
- Kurssin suorittaminen: laskuharjoitukset (10 p.) + ohjelmointiprojekti (20 p.) + koe (30 p.) = 60 p.
- Harjoitustehtävistä on merkittävä vähintään 20%. Maksimipisteet 10 pistettä saa merkitsemällä noin 80% harjoitustehtävistä.
- Ohjelmointiprojekti vaaditaan myös erilliskokeeseen osallistuvilta.
- Kurssimateriaali perustuu aikaisempiin J. Tarhion, E. Ukkosen, T. Hegedüksen ja V. Mäkisen laatimiin muistiinpanoihin sekä kirjallisuusluetteloissa mainittuihin kirjoihin. Suuri osa luennoitavista asioista löytyy kirjasta

  Crochemore & Rytter: *Jewels of Stringology*.
  World Scientific 2002.

Johdanto

- Merkkijono (engl. string) on yksinkertaisimpia, luonnollisimpia ja yleisimpiä tiedon talletusmuotoja
  - luonnollinen kieli, teksti
  - mikä tahansa tiedosto
  - DNA- ja proteiinisekvenssit
  - musiikki
- Merkkijonomenetelmät (stringologia, engl. stringology, string matching) on algoritmitutkimuksen erikoisala, jonka tutkii merkkijonoihin liittyviä algoritmeja ja tietorakenteita:
  - merkkijonohahmon etsintä tekstistä
  - merkkijonojen talletusrakenteet, indeksointi
  - säännönmukaisuuksien etsintä tekstistä
  - tekstin tiivistäminen
- Esimerkkiongelmia
  - Hahmonsovitus: Esiintyykö `ab` jonossa `cbbbbacbbbabcc`? Entä `a*b*a`, missä `*` on *jokerimerkki*, eli sen tilalla voi olla mikä tahansa merkki?
  - Muodosta hahmonsovitusautomaatti, loppuosa-automaatti, osajonoautomaatti.
  - Etsi/laske pisin toistuva osajono, erilaisten osajonojen määrä, aakkosjärjestyksessä ensimmäinen loppuosa, kahden merkkijonon editointietäisyys.
  - Etsi merkkijonojoukon pisin yhteinen osajono, lyhin yhteinen ylijono (superstring), pisin yhteinen alijono (subsequence)
- Sovellusalueita

- tiedon haku sisällön perusteella
- tekstitietokannat
- luonnollisen kielen analyysi
- biotekniikan tietojenkäsittely
- DNA-sekvensointi, Human Genome Project
- query-by-humming: hyräillyn melodian haku musiikkitietokannasta
- kuvankäsittely, hahmontunnistus

• Tyypillisiä piirteitä

- ongelman koko usein hyvin suuri ($|$Human Genome$|=3\cdot 10^9$) ==> tarvitaan nopeita algoritmeja
- algoritmit usein hyvin lyhyitä, suunnittelu vaatii pikkutarkkuutta
- tekniikoita: äärelliset automaatit, puut, triet, dynaaminen ohjelmointi, hajautus, bittirinnakkaisuus

• Kurssin sisällöstä.

- Kurssilla käsitellään keskeisimpiä alan tuloksia. Useimmat algoritmit on kuvattu myös alan oppikirjoissa.
- Esitetään historiallisesti merkittävät algoritmit.
- Esitetään laaja kirjo erilaisia tekniikoita.
- Pyritään esittämään mahdollisimman tehokkaita algoritmeja.
- Painopiste (aikavaativuuden) pahimman tapauksen analyysissa.

Luku 1

Tarkka hahmonsovitus

Ongelma.

Olkoon $S = S[0, n) = s_0 s_1 \cdots s_{n-1} \in \Sigma^*$ merkkijono (teksti) ja $P = P[0, m) = p_0 p_1 \cdots p_{m-1} \in \Sigma^*$ toinen merkkijono (hahmo) aakkostossa Σ, $|\Sigma| = \sigma$. Sisältääkö S osajononaan P:n, ts. onko S muotoa $S = S'PS''$ jollakin jonoilla S', S''?

Huom. Yleensä sovelluksissa $m \ll n$, ja n voi olla hyvin suuri.

Esim.

```
S: karjalainen  karjalainen
P: aine               aine
```

Triviaaliratkaisu.

Kokeillaan täsmääkö hahmo alkaen

tekstipositiosta 0: onko $s_0 = p_0, s_1 = p_1, \ldots, s_{m-1} = p_{m-1}$?

tekstipositiosta 1: onko $s_1 = p_0, s_2 = p_1, \ldots, s_m = p_{m-1}$?

kunnes löydetään positio, josta alkaen koko hahmo täsmää tai S päättyy.

```
karjal|aine|n
aine  |    |
 aine |    |
  aine|    |
   ain|e   |
    ai|ne  |
     a|ine |
      |aine|
```

Algoritmi 1.0.1

Syöte $S = s[0 \ldots n)$ teksti, $P = p[0 \ldots m)$ hahmo
Tuloste P:n ensimmäinen esiintymän alkukohta S:ssä

```
(1)  i := 0; j := 0;
(2)  while i < m and j < n do
(3)      if p[i] = s[j] then
(4)          i := i + 1; j := j + 1;
(5)      else
(6)          j := j − i + 1; i := 0;
(7)  if i = m then write j − m;
```

- Aikavaatimus: verrannollinen merkkivertailujen $p[i] = s[j]$ (rivi 3) kokonaislukumäärään

  1. Pahin tapaus: $P = aaa \cdots ab$ ja $S = aa \cdots a$. Vertailujen lukumäärä ja samalla aikavaatimus on siis $O(mn)$.
  2. Keskimääräinen käyttäytyminen: jos kaikki merkit ovat yhtä todennäköisiä S:ssä ja P:ssä, voidaan osoittaa, että aikavaatimus on $O(n)$.

- Tilavaatimus: $O(1)$

- Toimii kohtalaisen hyvin käytännössä.

1.1 Knuth-Morris-Pratt-algoritmi

D. Knuth, J. Morris & V. Pratt: Fast pattern matching in strings. *SIAM Journal on Computing* 6:323–350, 1977.

Kehitetään seuraavaksi eräänlainen äärelliseen automaattiin perustuva menetelmä jonka aikavaatimus on $O(n)$. Idea:

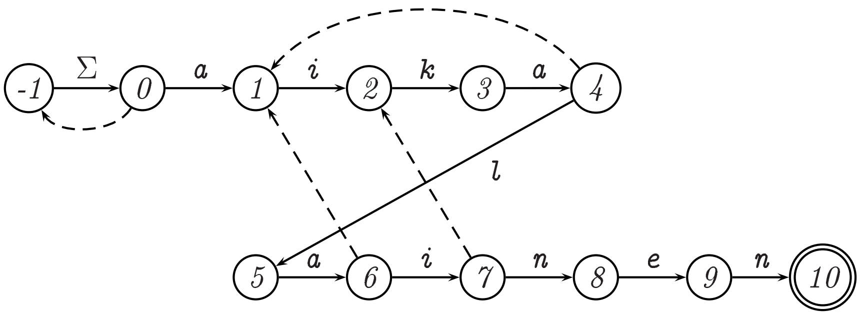
- vertailukohta *ei koskaan* siirry *taaksepäin* tekstissä.

- jos vertailun $s_j = p_i$ tulos on epätäsmäys, tiedetään kuitenkin, että $s_{j-i} = p_0$, $s_{j-i+1} = p_1$, $\ldots$, $s_{j-1} = p_{i-1}$. Nyt voidaan hahmoa siirtää oikealle enemmänkin kuin yksi askel, nimittäin suoraan ensimmäiseen sellaiseen kohtaan, että s_j:stä vasemmalle tuleva hahmon osa täsmää.

S: `aikalaikalainen`
P: `aikalai`<u>`n`</u>`en`
`.....aikalainen`
`5 askeleen siirto`

- Oletetaan, että $p_0, \ldots, p_{i-1}$ täsmäävät, mutta $p_i \neq s_j$.
- Määrätään jonon $P[0 \ldots i)$ pisin aito alkuosa, joka on myös sen loppuosa; olkoon se $P[0 \ldots k) = P[i-k \ldots i)$.
- Seuraavaksi verrataan $p_k = s_j$ eli hahmoa siirretään $i-k$ askelta oikealle, mutta tekstissä ei edetä.

Esimerkki 1.1.1 *Merkkijonosta* ***aikalainen*** *muodostettu jononsovitus-automaatti.*



KMP-algoritmin toiminta

- Hahmon P esiprosessointi: lasketaan korjaussiirtymät, jotka esitetään taulukkona $fail[0 \ldots m]$
- Tekstin selaus käyttäen
  - varsinaisia siirtymiä: edetään yksi askel oikealle P:ssä ja S:ssä,
  - korjaussiirtymä: jos varsinainen siirtymä ei onnistu (vertailu $s[j] = p[i]$ epäonnistuu) seurataan korjaussiirtymää $i = fail[i]$ ja verrataan seuraavaksi $s[j] = p[i]$.

Algoritmi 1.1.2 *Knuth-Morris-Pratt*

Syöte: $S = s[0 \dots n)$ teksti, $P = p[0 \dots m)$ hahmo
Tuloste; pienin i s.e. $s[j \dots j+m) = p[1 \dots m)$, tai jos tällaista i ei ole, n

Initfail; (* korjaussiirtymien laskenta Alg. 1.1.3:llä *)
i:=0; j:=0;
repeat
    (* invariantti: $p[0 \dots i) = s[j-i \dots j)$ *)
    if $i = -1$ **or** $p[i] = s[j]$ **then**
        $i := i+1; j := j+1;$
    else $i := fail[i]$
until $i = m$ **or** $j = n$;
if $i = m$ **then return** $j - m$
else return n

Korjaussiirtymät

- Taulukko $fail[0 \dots m]$ vastaa jononsovitus-automaatin korjaussiirtymiä.
- Määritelmä: Jono x on jonon y *alkuosa* (prefix) jos $y = xu$ jollakin jonolla u ja *loppuosa* (suffix) jos $y = vx$ jollakin jonolla v. Alkuosa on *aito*, jos $u \neq \epsilon$ (ϵ tarkoittaa tyhjää jonoa) ja loppuosa on *aito* jos $v \neq \epsilon$.
- $fail$-arvoilla tulee olla seuraava ominaisuus:
  - $fail[i] = k$, kun $P[0 \dots k)$ on jonon $P[0 \dots i)$ pisin aito alkuosa, joka on myös sen loppuosa
  - $fail[0] = -1$

Fail-arvojen laskenta

- Määritelmän mukaan $fail[0] = -1$.
- Oletetaan, että $fail[0 \dots i]$ on laskettu.
  - Olkoon $fail[i] = k$, siis $P[0 \dots k) = P[i-k \dots i)$
  - jos $p_i = p_k$ niin $fail[i+1] = k+1$, koska $P[0 \dots k+1) = P[i-k \dots k+1)$

  - jos $p_i \neq p_k$, niin määrätään pienin r, jolla $p_i = p_u$ ja $fail^r[i] = u$ (eli $fail[fail[\dots fail[i]\dots] = u$), jolloin $fail[i+1] = u+1$, tai $fail^r[i] = -1$, jolloin $fail[i+1] = 0$.

Algoritmi 1.1.3 *Initfail*

Syöte: $P = p[0\dots m)$ hahmo
Tuloste: taulukko $fail[0\dots m]$ hahmolle P
(1) $k := -1$; $i := 0$; $fail[0] := -1$;
(2) **repeat**
(3)   **if** $i = -1$ **or** $p[i] = p[k]$ **then**
(4) $i := i+1; k := k+1; fail[i] := k$
(5)   **else** $k := fail[k]$
(6) **until** $i = m$

Lause 1.1.4 *Algoritmi 1.1.3 vaatii ajan $O(m)$.*

Todistus. Aikavaatimus on verrannollinen repeat-lauseen ehtolauseen (3-5) suorituskertojen lukumäärään k. Kullakin kerralla suoritetaan joko then-haara (4) tai else-haara (5):

- then-haara
  - kasvattaa aina i:tä yhdellä
  - i ei missään pienene
  - alussa $i = 0$ ja lopussa $i = m$
  - ==> then-haara suoritetaan m kertaa
- else-haara
  - pienentää k:tä aidosti (koska $fail[k] < k$)
  - kasvattaa siten $i - k$:tä
  - $i - k$ ei koskaan pienene
  - alussa $i - k = 1$, lopussa $i - k \leq m + 1$
  - ==> else-haara suoritetaan korkeintaan m kertaa
- ==> $k \leq 2m = O(m)$

□

Lause 1.1.5 *Algoritmi 1.1.2 vaatii ajan $O(m + n)$.*

Todistus. Esiprosessointi (Alg. 1.1.3) vaatii ajan $O(m)$ (Lause 1.1.4). Tekstin selaus (Algoritmin 1.1.2 rivit 3–7) vaatii ajan $O(n)$. Tämä osoitetaan kuten Lause 1.1.4. □

1.2 Boyer-Moore-algoritmi

R. Boyer & S. Moore: A fast string searching algorithm.
Communications of the ACM 20: 762–772, 1977.

Boyer-Moore-algoritmin (alkuperäinen) idea:

- Hahmoa siirretään vasemmalta oikealle (kuten edellä).
- Jokaisessa pysähdyskohdassa merkkivertailut aloitetaan P:n lopusta ja edetään alkuunpäin (siis oikealta vasemmalle).
- Jos vuorossa oleva merkkipari ei täsmää, käytetään jälleen erästä siirtofunktiota, joka siirtää hahmoa $1 \ldots m$ askelta oikealle.
- Algoritmi tutkii parhaassa tapauksessa vain joka m:nnen S:n merkin!

Esimerkki 1.2.1
$P =$ *at-that*, $S =$ *which-finally-halts-at-that-point*:

```
which-finally-halts---at-that-point
      ?      ?  ?=   ?==
at-that      |  ||   |||        hahmon ensimmäinen kohdistus
       at-that  ||   |||        siirto tähän, koska f ei esiinny P:ssä
           at-that   |||        siirto tähän, koska nyt - täsmää
                 at-that        siirto tähän, koska l ei esiinny P:ssä
                      at-that   siirto tähän, koska nyt at täsmää;
                                havaitaan, että koko hahmo täsmää.
```

Tarkastellaan tilannetta, jossa verrataan merkkejä p_i ja s_j ja tuloksena on epätäsmäys. Siirron määräämisessä käytetään kahta heuristiikkaa:

1. *Esiintymäheuristiikka:* Konfliktin aiheuttaneen S:n merkin s_j kohdalle tulee siirron jälkeen P:n lopusta lukien ensimmäinen merkki, joka täsmää. Jos P ei sisällä s_j:tä, siirron pituus on m.

2. *Sovitusheuristiikka:* Hahmoa siirretään oikealle, kunnes tutkittu S:n osa $s[j+1\ldots j+m-i) = p[i+1\ldots m)$ täsmää P:n kohdalle tulevaan osaan. Lisäksi konfliktin aiheuttaneen S:n merkin s_j kohdalle täytyy siirron jälkeen tulla jokin muu merkki kuin p_i.

Lopullinen siirto on maksimi heuristiikkojen 1 ja 2 antamista siirroista.

Algoritmi 1.2.2 *Boyer-Moore*

Syöte: $S = s[1..n)$ teksti, $P = p[1..m)$ hahmo
Tuloste: P:n ensimmäisen esiintymän alkukohta S:ssä
(1) init; (* taulukkojen δ_1 ja δ_2 alustus *)
(2) $j := m-1$;
(3) **while** $j < n$ **do**
(4) $\quad i := m-1$;
(5) $\quad$ **while** $i \geq 0$ **and** $p[i] = s[j]$ **do**
(6) $\quad\quad i := i-1;\ j := j-1$;
(7) $\quad$ **if** $i < 0$ **then return** $j+1$;
(8) $\quad j := j + \max(\delta_1[s[j]], \delta_2[i])$

Rivillä 8 $s[j]$ on konfliktimerkki. Taulukko δ_1 antaa heuristiikan 1 mukaisen lukukohdan siirron:
$\delta_1[a] = \min\{t \mid t = m \text{ tai } a = p[m-1-t])\}$.

$S:$ `halts---at-that`
$P:$ `   at-that` $\quad p[m-1-4] =$ `-` $=> \delta_1[s] = 4$
`      at-that`

Taulukko δ_2 antaa heuristiikan 2 mukaisen lukukohdan siirron:
$\delta_2[i] = m-1-i+\min(\{t \mid t > i \text{ and } P[0\ldots m-t) = P[t\ldots m)\} \cup$
$\{t \mid 1 \leq t \leq i \text{ and } P[i+1\ldots m) = P[i+1-t\ldots m-t) \text{ and } P[i] \neq P[i-t]\})$

$S:$ `halts---at-that`
$P:$ `   at-that` $\quad t = 5 => \delta_2[4] = 7-1-4+5 = 7$
`        at-that`

- Taulukko δ_1 on helppo laskea. Aikavaatimus: $O(\sigma + m)$.
- Taulukko δ_2 lasketaan KMP-esiprosessoinnin (Alg. 1.1.3) tapaan. Aikavaatimus: $O(m)$.
- Taulukoissa δ_1 ja δ_2 on yhteensä $\sigma + m + 1$ alkiota.

BM-algoritmin esiprosessointiaika on $O(\sigma+m)$. Tekstin selausaika pahimmassa tapauksessa $O(n)$ ja parhaassa tapauksessa $O(n/m)$. BM-algoritmin analyysi on varsin mutkikas.

BM-algoritmista on kymmeniä muunnelmia, jotka pyrkivät korjaamaan sen heikkouksia. Tässä joitakin:

- Kaikkien esiintymien löytäminen
  - BM-algoritmi on helppo yleistää etsimään kaikki hahmon esiintymät, mutta selausaika kasvaa $\mathcal{O}(n + rm)$:ään, missä r on löydettyjen esiintymien lukumäärä.
  - Boyer-Moore-Galil -algoritmi [1] löytää kaikki esiintymät ajassa $\mathcal{O}(n)$.
  - BM-algoritmi hahmoa siirtäessään unohtaa kaiken aiemmista täsmäyksistä.
  - BMG-algoritmi pitää muistissa mahdollisesti jo täsmääväksi tiedetyn hahmon alkuosan pituuden.
- Keskimääräinen aikavaatimus
  - Yao [4] on näyttänyt, että alaraja ongelman keskimääräiselle aikavaatimukselle on $\Omega((n/m)\log_\sigma m)$.
  - BM-algoritmi ei pääse tähän, koska se tekee liian lyhyitä siirtymiä.
  - Lecroqin esittämä variaatio [3] on optimaalinen keskimääräisessä tapauksessa.
  - Se ei lopeta täsmäystä vielä, kun teksti ei enää täsmää hahmon loppuosaan, vaan vasta, kun teksti ei enää täsmää mihinkään hahmon osajonoon.
- Käytännön suorituskyky
  - Sovitusheuristiikka on keskeinen teoreettiselle analyysille, mutta useimmissa käytännön tilanteissa esiintymäheuristiikka ratkaisee nopeuden.
  - Boyer-Moore-Horspool -algoritmi [2] käyttää vain esiintymäheuristiikkaa, mutta ei epätäsmäyksen aiheuttaneen merkin perusteella vaan hahmon viimeisen merkin kohdalla olevan tekstin merkin perusteella.
  - Ohitussilmukka (skip loop): Etsitään hahmon viimeistä merkkiä nopeassa silmukassa.

1.3 Karp-Rabin -algoritmi

R. Karp & M. Rabin: Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development* 31:249–260, 1987.

Idea

Käytetään hajautusfunktiota H: jonot $\rightarrow$ kokonaisluvut. Lasketaan $H(P)$ ja tutkitaan jokaisella $j = 0, \ldots, n-m$, onko $H(P) = H(S[j \ldots j+m))$.

Jos $H(P) \neq H(s[j \ldots j+m))$, P ei voi täsmätä $S[j \ldots j+m)$:n kanssa. Muuten tarkastetaan, onko todella $P = S[j \ldots j+m)$. Koska H sallii yhteentörmäykset (H ei ole injektio), saattaa olla $P \neq S[j \ldots j+m-1]$.

Karp ja Rabin (1987) käyttävät funktiota H, jolla on ominaisuudet:

- $H(S[j+1 \ldots j+m+1))$ on laskettavissa nopeasti arvosta $H(S[j \ldots j+m))$ ja merkistä $S[j+m]$.
- Yhteentörmäykset ovat harvinaisia.

Hajautusfunktion H määrittely

- Tulkitaan aakkoston Σ merkit numeroina $0, 1, \ldots, \sigma-1$.
- Merkkijono $A = a_0 \cdots a_{k-1}$ esittää σ-järjestelmän kokonaislukua $v(A) = a_0\sigma^{k-1} + a_1\sigma^{k-2} + \cdots + a_{k-1}\sigma + a_{k-1}$.
- H:n arvoalueeksi valitaan $\{0, 1, \ldots, q-1\}$ missä q on alkuluku.
- Asetetaan $H(A) = H_q(A) = v(A)\ mod\ q$. ($mod$ on jakojäännös, esim. $6\ mod\ 3 = 0$, $7\ mod\ 3 = 1$, $8\ mod\ 3 = 2$, $9\ mod\ 3 = 0$.)
- Tarkastellaan H_q:n laskemista peräkkäisissä tekstipositioissa $S^j = s[j \ldots j+m)$ ja $S^{j+1} = s[j+1 \ldots j+m+1)$:
  - Havaitaan: $v(S^{j+1}) = (v(S^j) - s[j] \cdot \sigma^{m-1}) \cdot \sigma + s[j+m]$.
  - *mod*-aritmetiikassa
    $(a+b)\ mod\ q = (a\ mod\ q + b\ mod\ q)\ mod\ q$
    $(a \cdot b)\ mod\ q = ((a\ mod\ q) \cdot (b\ mod\ q))\ mod\ q$

– siten

$$\begin{aligned} H_q(S^j) &= v(S^{j+1}) \; mod \; q \\ &= ((v(S^j) \; mod \; q - s[j] \cdot \sigma^{m-1} \; mod \; q) \cdot \sigma \; mod \; q + \\ &\quad s[j+m]) \; mod \; q \\ &= (H_q(S^j) - s[j] \cdot \sigma^{m-1}) \cdot \sigma + s[j+m], \end{aligned}$$

missä kaikki aritmeettiset operaatiot suoritetaan mod q -aritmetiikassa.

– Siis $H_q(S^{j+1})$ saadaan lasketuksi $H_q(S^j)$:stä 4:llä mod q -aritmetiikan operaatiolla. Tämä vie vakioajan, jos q on valittu niin, että kukin välitulos mahtuu yhteen sanaan.

Algoritmi 1.3.1 *Karp-Rabin*

Syöte: $S = s[1 \ldots n]$ teksti, $P = p[1 \ldots m]$ hahmo
Tuloste: kaikkien P:n esiintymien alkukohdat S:ssä
(1) $\sigma := 32$;
(2) $q := 4194301$; (* suurin alkuluku $< 2^32/\sigma^2$ *)
(3) cm $:= \sigma^{m-1} \; mod \; q$;
(4) $hp := 0; hs := 0$;
(5) **for** $i := 0$ **to** $m-1$ **do** $hp := (hp \cdot \sigma + p[i]) \; mod \; q$; (* $hp = H(P)$ *)
(6) **for** $j := 0$ **to** $m-1$ **do** $hs := (hs \cdot \sigma + s[j]) \; mod \; q$; (* $hs = H(S^0)$ *)
(7) **for** $j := 0$ to $n-m-1$ **do begin**
(8) **if** $hp = hs$ **then if** $P = S[j \ldots j+m)$ **then** write i;
(9) $hs := ((hs - s[j] \cdot \text{cm}) \cdot \sigma + s[j+m]) \; mod \; q$;
(10) **if** $hp = hs$ **then if** $P = S[n-m \ldots n)$ **then** write $n-m$;

Huomautuksia Algoritmista 1.3.1:

- Algoritmi toimii vakiotilassa.
- Jotta algoritmi toimisi nopeasti, on edullista, jos kukin H:n arvo mahtuu yhteen sanaan. Jos sanan pituus on 32 bittiä, kannattaa valita q:ksi suurin alkuluku $< 2^{32}/\sigma^2$.
- Aakkoston kokona kannattaa käyttää 2:n potenssia, jolloin σ:lla kertomiset voidaan toteuttaa sivuttaissiirtoina (shift).
- Rivin 9 lauseke $hs - s[j] \cdot \text{cm}$ pitää kirjoittaa jossain toteutusympäristöissä termillä $\sigma \cdot q$ lisättynä eli muodossa $hs + (\sigma - 1) \cdot q - s[j] \cdot \text{cm}$, jotta *mod* otettaisiin varmasti positiivisesta arvosta.

- Vertailuissa $P = S^0$ ja $P = S^j$ riveillä 8 ja 10 vertaillaan m-merkkisiä jonoja. Ne toteutetaan esim. triviaalialgoritmilla.

Sanomme

- Väärä täsmäys (false match): $H(P) = H(S^j)$, mutta $P \neq S^j$.
- Oikea täsmäys (proper match): $H(P) = H(S^j)$ ja $P = S^j$.

Algoritmin 1.3.1 aikavaatimus on selvästi muotoa $O(m + n + fm + hm)$, missä f on väärien ja h oikeiden täsmäysten lukumäärä. Pahimmassa tapauksessa tämä on $O(mn)$. Keskimääräisessä tapauksessa (tietyin oletuksin) f ja h ovat pieniä, joten keskimääräisessä tapauksessa aikavaatimus on $O(m+n)$.

Karp ja Rabin tarkastelevat myös universaaliin hajautukseen perustuvia versioita algoritmista:

- Valitaan q *satunnaisesti* lukujoukosta Q.
- Q:n luvut voivat olla pieniäkin, eikä niiden tarvitse olla kokonaislukuja.
- Väärän täsmäyksen todennäköisyys on erittäin pieni syötteestä riippumatta (universaalius).
- q voidaan myös vaihtaa aina väärän täsmäyksen sattuessa.

1.4 Eliminointimenetelmä

U. Vishkin: Deterministic sampling — a new technique for fast pattern matching. *SIAM Journal of Computing* 20(1):22–40, 1991.

- *Kandidaatti*: tekstin positio, josta hahmon esiintymä voi alkaa.
- Idea:
  - Asetetaan kaikki tekstin positiot kandidaateiksi.
  - Karsitaan kandidaatteja tutkimalla tekstiä.
  - Operaatio: *parivertailu* (duel, kaksintaistelu).
  - Jäljelle jääneitä kandidaatteja vastaavat esiintymät tutkitaan tarkasti.
- Käyttöalueet: rinnakkaisalgoritmit ja kaksiulotteinen merkkijononsovitus.

Kirjallisuutta

[1] Zvi Galil. On improving the worst-case running time of the boyer–moore string-searching algorithm. *Communications of the ACM*, 22:505–508, 1979.

[2] N. Horspool. Practical fast searching in strings. *Software-Practice and Experience*, 10:501–506, 1980.

[3] T. Lecroq. A variation of the boyer-moore algorithm. *Theoretical Computer Science*, 92:119–144, 1992.

[4] A. C.-C. Yao. The complexity of pattern matching for a random string. *SIAM Journal on Computing*, 8:368–387, 1979.