58093 Merkkijonomenetelmät (6 op), syksy 2008

Juha Kärkkäinen[1]

4. syyskuuta 2008

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

Yleistä kurssista

- Kurssi on valinnainen syventävä kurssi, joka soveltuu erityisesti *algoritmien (ja koneoppimisen) erikoistumislinjan* sekä *bioinformatiikan* opiskelijoille.
  - Soveltuu algoritmien erikoistumislinjan pakolliseksi syventäväksi kurssiksi (2005-2008 tutkintovaatimukset). Laajuus on supistunut 8 op:stä 6 op:een, mutta "puuttuvat" 2 op:tä voi korvata valinnaisilla linjakohtaisilla kursseilla.
- Esitiedoiksi oletetaan kurssien *Tietorakenteet* sekä *Laskennan mallit* (tai sen edeltäjän *Ohjelmoinnin ja laskennan perusmallit*) tiedot.
  Lisäksi hyödyllisiä ovat kurssit *Laskennan vaativuus* sekä *Algoritmien suunnittelu (ja analyysi)*.
- Periodin II kurssilla *Tiedon tiivistämisen tekniikat* käytetään tämän kurssin tekniikoita.
- Luennot: Juha Kärkkäinen | 02.09.-09.10. | ti, to 12-14 B222
  - Luennot ti 16.9., to 18.9. ja ti 23.9. peruutettu. Korvaavat luennot ovat ma 8.9., ma 15.9. ja ma 29.9. 12-14 B222.
- Harjoitukset: Jarkko Toivonen | 09.09.-07.10. | ti 10-12 CK111
- Kurssiin kuuluu ohjelmointiprojekti, jossa toteutetaan kurssin aihepiiriin kuuluva algoritmi ja testataan sitä sopivalla syöteaineistolla. Tulokset raportoidaan postereina.
  - Aiheiden valinta: 30.09.
  - Projektisuunnitelman palautus: 10.10.
  - Posteriesittely: 25.11.?? (periodin II 5. viikko)
- 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, 2003.

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, string processing algorithms) 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 ($|\text{Human Genome}|=3\cdot 10^9$) ==> 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 $S = S[0 \ldots n) = s_0 s_1 \cdots s_{n-1} \in \Sigma^n \subset \Sigma^*$ merkkijono (teksti) ja $P = P[0 \ldots m) = p_0 p_1 \cdots p_{m-1} \in \Sigma^m \subset \Sigma^*$ toinen merkkijono (hahmo) aakkostossa Σ, $|\Sigma| = \sigma$. Sisältääkö S osajononaan P:n, ts. onko S muotoa $S = S'PS''$ joillakin 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 ensimmäisen 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 output 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 $\mathcal{O}(mn)$.
  2. Keskimääräinen käyttäytyminen: jos kaikki merkit ovat yhtä todennäköisiä S:ssä ja P:ssä, voidaan osoittaa, että aikavaatimus on $\mathcal{O}(n)$.
- Tilavaatimus: $\mathcal{O}(1)$
- 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.

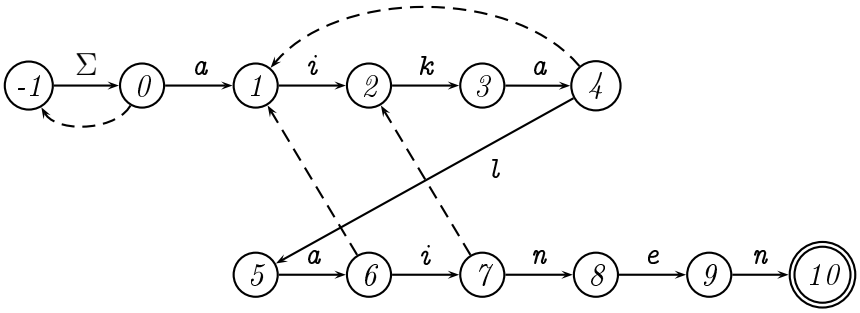
Kehitetään seuraavaksi eräänlainen äärelliseen automaattiin perustuva menetelmä jonka aikavaatimus on $\mathcal{O}(n)$. Idea:

- Hyödynnetään jo tehtyjä täsmäyksiä.
- 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: aikalainen
   .....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 liikuta.
- Vertailukohta tekstissä *ei koskaan* siirry *taaksepäin* $\Rightarrow$ automaatti.

Esimerkki 1.1.1 *Merkkijonosta* `aikalainen` *muodostettu jononsovitusautomaatti ja sen korjaussiirtymätaulukko.*



i	0	1	2	3	4	5	6	7	8	9	10
$fail[i]$	-1	0	0	0	1	0	1	2	0	0	0

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

```
(1) Initfail; (* korjaussiirtymien laskenta Alg. 1.1.3:llä *)
(2) i:=0; j:=0;
(3) while i < m and j < n do
(4)       (* invariantti: P[0...i) = S[j-i...j) *)
(5)       if i = -1 or p[i] = s[j] then
(6)             i := i+1; j := j+1;
(7)       else i := fail[i]
(8) if i = m then return j-m
(9) else return n
```

Korjaussiirtymät

- Taulukko $fail[0 \ldots m]$ 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 $P[0 \ldots k)$ on jonon $P[0 \ldots i)$ pisin aito alkuosa, joka on myös sen loppuosa
  - $fail[0] = -1$

Fail-arvojen laskenta

- Määritelmän mukaan $fail[0] = -1$.
- Oletetaan, että $fail[0 \ldots i]$ on laskettu.
  - Olkoon $fail[i] = k$, siis $P[0 \ldots k) = P[i-k \ldots i)$
  - jos $p_i = p_k$ niin $fail[i+1] = k+1$, koska $P[0 \ldots k+1) = P[i-k \ldots k+1)$
  - jos $p_i \neq p_k$, niin määrätään *pienin* r, jolla $p_i = p_u$ ja $fail^r[i] = u$ (eli $fail[fail[\ldots fail[i]\ldots] = u$), jolloin $fail[i+1] = u+1$, tai $fail^r[i] = -1$, jolloin $fail[i+1] = 0$.

Algoritmi 1.1.3 *Initfail*

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

Lause 1.1.4 *Algoritmi 1.1.3 vaatii ajan $\mathcal{O}(m)$.*

Todistus. Aikavaatimus on verrannollinen while-silmukan suorituskertojen lukumäärään. 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:ta 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
- yhteensä $\leq 2m = \mathcal{O}(m)$

□

Lause 1.1.5 *Algoritmi 1.1.2 vaatii ajan $\mathcal{O}(m + n)$.*

Todistus. Esiprosessointi (Alg. 1.1.3) vaatii ajan $\mathcal{O}(m)$ (Lause 1.1.4). Tekstin selaus (Algoritmin 1.1.2 rivit 3–7) vaatii ajan $\mathcal{O}(n)$. 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 $1 - -m$ 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 $i + 1$.
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: $S = S[0..n)$ teksti, $P = P[0..m)$ 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      p[m-1-4] = - => δ1[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      t = 5 => δ2[4] = 7 - 1 - 4 + 5 = 7
             at-that
```

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

BM-algoritmin esiprosessointiaika on $\mathcal{O}(\sigma+m)$. Tekstin selausaika pahimmassa tapauksessa $\mathcal{O}(n)$ ja parhaassa tapauksessa $\mathcal{O}(n/m)$. 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 (C. ACM 22(9):505–508, 1979) 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 (SIAM J. Comp., 8(3):368–387, 1979) on osoittanut, 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 (Theoretical Computer Science, 92:119–144, 1992). 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.
  - Horspool -algoritmi (Software—Practice and Experience, 10:501–506, 1980) 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 Shift-Or-algoritmi

R. Baeza-Yates & G. Gonnet: A new approach to text searching. *Communications of ACM* 35(10):74–82, 1992.

Olkoon w tietokoneen sananpituus. Oletetaan aluksi, että $|P| = m \leq w$.

- Ylläpidetään m-bittistä bittivektoria E, joka esittää haun tilanteen kullakin hetkellä. Bitti $E.i = 0$, jos $p[0 \ldots i]$ täsmää tähän asti luetun tekstin loppuun. Eli, kun tekstistä on luettu $S[0 \ldots j]$, $E.i = 0$, jos $P[0 \ldots i] = S[j - i \ldots j]$, muuten $E.i = 1$.
- Esiintymä on löytynyt, kun bitti $E.(m-1) = 0$.
- Tilannetta päivitetään kussakin tekstin kohdassa käyttämällä kullekin merkille $a \in \Sigma$ laskettuja bittivektoreita $T[a]$, joille pätee bitti $T[a].i = 0$, jos $p[i] = a$, muuten $T[a].i = 1$.
- Päivitys toteutetaan kahdessa vaiheessa:
  (1) E:n bittejä siirretään askel vasemmalle ($E' := E << 1$)
  (2) Suoritetaan bittivektorien E' ja $T[s[j]]$ or-operaatio ($E' \mid\mid T[s[j]]$).

Algoritmi 1.3.1 *Shift-Or*

Syöte: $S, P, m \leq w$
Tuloste: P:n esiintymien loppukohdat

(1) **for** $a \in \Sigma$ **do** $T[a] := 2^m - 1$; (* $T[a] := 111...11$ *)
(2) **for** $i := 0$ **to** $m - 1$ **do** $T[p[i]] := T[p[i]] - 2^i$ (* $T[a].i := 0$ *)
(3) $E := 2^m - 1$;
(4) **for** $j := 0$ **to** $n - 1$ **do**
(5) $E := (E << 1) \mid\mid T[s[j]]$
(6) **if** $E.(m-1) = 0$ **then** write j

Esimerkki 1.3.2 $P =$ `assi`, $S =$ `apassit`, *bittivektorit pystysuorassa*

`apuvektorit`

	a	*i*	*p*	*s*	*t*
a	0	1	1	1	1
s	1	1	1	0	1
s	1	1	1	0	1
i	1	0	1	1	1

`sovituksen vaiheet`

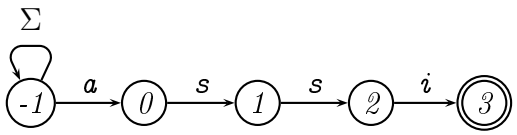
	a	*p*	*a*	*s*	*s*	*i*	*t*
1	0	1	0	1	1	1	1
1	1	1	1	0	1	1	1
1	1	1	1	1	0	1	1
1	1	1	1	1	1	0	1

Varaamalla E:lle useita sanoja, voidaan käsitellä pitempiä hahmoja. Algoritmin ominaisuuksia:

- Selauksen (rivit 4-6) aikavaatimus: $O(\lceil m/w \rceil n)$.
- Alustuksen (rivit 1-2) aikavaatimus: $O(\lceil m/w \rceil \sigma + m)$.
- Tilavaatimus: $O(\lceil m/w \rceil \sigma)$.

Shift-or-algoritmin voidaan myös ajatella simuloivan epädeterminististä hahmonsovitusautomaattia. Bittivektori E on automaatin tilavektori:
Kun $S[0 \ldots j]$ on selattu, $E.i = 0$ joss automaatissa on polku, joka johtaa tilaan i syötteellä $S[0 \ldots j]$.

Esimerkki 1.3.3 *Hahmon assi epädeterministinen hahmonsovitusautomaatti.*



1.4 Karp-Rabin -algoritmi

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

Idea

- Merkitään $S^j = S[j \ldots j + m)$.
- Käytetään hajautusfunktiota H: merkkijonot $\rightarrow$ kokonaisluvut. Lasketaan $H(P)$ ja tutkitaan jokaisella $j = 0, \ldots, n - m$, onko $H(P) = H(S^j)$.
- Jos $H(P) \neq H(S^j)$, niin täytyy olla $P \neq S^j$.
- Jos $H(P) = H(S^j)$, tarkastetaan, onko todella $P = S^j$. Koska H sallii yhteentörmäykset (H ei ole injektio), saattaa olla $P \neq S^j$.

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

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

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 $v(A) = a_0\sigma^{k-1} + a_1\sigma^{k-2} + \cdots + a_{k-1}\sigma + a_{k-1}$.
- v toteuttaa ehdot:
  - ei yhteentörmäyksiä
  - $v(S^{j+1}) = (v(S^j) - s[j] \cdot \sigma^{m-1}) \cdot \sigma + s[j+m]$.
- $v(A)$ voi olla kuitenkin liian suuri luku tehokkaasti käsiteltäväksi.
- H:n arvoalueeksi valitaan $\{0, 1, \ldots, q-1\}$ missä q on alkuluku.
- Asetetaan $H(A) = H_q(A) = v(A) \bmod q$. (*mod* on jakojäännös, esim. $6 \bmod 3 = 0$, $7 \bmod 3 = 1$, $8 \bmod 3 = 2$, $9 \bmod 3 = 0$.)
  - *mod*-aritmetiikassa
    $(a + b) \bmod q = (a \bmod q + b \bmod q) \bmod q$
    $(a \cdot b) \bmod q = ((a \bmod q) \cdot (b \bmod q)) \bmod q$
  - siten

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

  - 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 1.4.1 *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) $\mathrm{cm} := \sigma^{m-1} \ mod \ q$;
(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**
(8) **if** $hp = hs$ **then if** $P = S[j \ldots j+m)$ **then** write j;
(9) $hs := ((hs - s[j] \cdot \mathrm{cm}) \cdot \sigma + s[j+m]) \bmod q$;
(10) **if** $hp = hs$ **then if** $P = S[n-m \ldots n)$ **then** write $n-m$;

Huomautuksia Algoritmista 1.4.1:

- Algoritmi toimii vakiotilassa.
- Jotta algoritmi toimisi nopeasti, on edullista, jos kukin H:n arvo ja välitulokset mahtuvat 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 $hs - s[j] \cdot \mathrm{cm}$ pitää kirjoittaa jossain toteutusympäristöissä termillä $(\sigma-1) \cdot q$ lisättynä eli muodossa $hs + (\sigma-1) \cdot q - s[j] \cdot \mathrm{cm}$, jotta *mod* otettaisiin varmasti positiivisesta arvosta.
- Vertailuissa $P = S^j$ ja $P = S^{n-m}$ 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 (true match): $H(P) = H(S^j)$ ja $P = S^j$.

Algoritmin 1.4.1 aikavaatimus on selvästi muotoa $\mathcal{O}(m + n + fm + hm)$, missä f on väärien ja h oikeiden täsmäysten lukumäärä. Pahimmassa tapauksessa tämä on $\mathcal{O}(mn)$. Keskimääräisessä tapauksessa (tietyin oletuksin) f ja h ovat pieniä ja väärien täsmäysten tarkistaminen on nopeaa, joten keskimääräisessä tapauksessa aikavaatimus on $\mathcal{O}(m + n)$.