2.5 Seulontamenetelmät

- Edellä on kehitetty algoritmeja, joiden pahimman tapauksen aikavaativuudet ovat $\mathcal{O}(mn)$, $\mathcal{O}(kn)$, $\mathcal{O}(\lceil m/w \rceil n)$, tms.
- Monissa sovelluksissa ylläolevat aikavaativuudet ovat vielä liian suuria; tarvetta on lineaarisille tai jopa alilineaarisille ratkaisuille.
- *Seulontamenetelmillä* voidaan saavuttaa nämä tavoitteet odotusarvoisessa tapauksessa, kun sallittu virheiden määrä k on tarpeeksi pieni.
- Seulontamenetelmien yleinen kehikko:
  - *Vaihe 1*: Käytetään algoritmia F, jonka aikavaatimus on $\mathcal{O}(f(m,n,k) + |Pocc|)$, *seulomaan* tekstistä joukko $Pocc = \{j\}$ *mahdollisia* esiintymien loppukohtia.
  - *Vaihe 2*: Käytetään algoritmia G, jonka aikavaatimus on $\mathcal{O}(g(m,n,k,|Pocc|))$, *tarkistamaan* onko kukin $j \in Pocc$ *todellisen esiintymän* loppukohta.
- Tavoitteena on valita F ja G siten, että kokonaissuoritusaika $\mathcal{O}(f(m,n,k) + |Pocc| + g(m,n,k,|Pocc|))$ on mahdollisimman pieni.
  - Käytännössä algoritmi G on esim. dynaaminen ohjelmointi tai Myersin algoritmi.
  - Jäljelle jää valita algoritmi F siten, että se toimii mahdollisimman nopeasti, mutta samalla tulostaa mahdollisimman vähän *vääriä* esiintymäkandidaatteja.

Odotusarvoisesti lineaariaikainen seulontamenetelmä

R. Baeza-Yates ja C. Perleberg. Fast and Practical Approximate String Matching. *Information Processing Letters*, 59:21–27, 1996.

- Olkoon P hahmo, S teksti, ja k yläraja virheiden määrälle.
- Etsitään siis kaikkia S:n positioita j, joihin päättyy jokin S:n osajono $S_{j'\ldots j}$ siten, että editointietäisyys $D(P, S_{j'\ldots j}) \leq k$.
- Johdetaan ongelman ratkaisemiseksi yksinkertainen seulontamenetelmä, joka perustuu seuraavaan lemmaan:

Lemma 2.5.1 *Jos hahmo P pilkotaan $k+1$:een palaseen $P = P^1P^2\cdots P^{k+1}$, niin ainakin yksi palasista löytyy muuttumattomana minkä tahansa likimääräisen esiintymän sisältä.*

Todistus. Kukin muutos, lisäys tai poisto voi muuttaa enintään yhtä palasista. Siten k editointioperaatiota voi muuttaa enintään k:ta palasta. □

- Käytetään Lemmaa 2.5.1 seuraavasti:
  - Vaihe 1: Etsitään kaikkien $k+1$:n palasen tarkkoja esiintymiä tekstistä Aho-Corasick automaatilla.
  - Vaihe 2: Tarkistetaan dynaamisella ohjelmoinnilla vaiheessa 1 löytyneet mahdolliset esiintymäkohdat.
- Algoritmin odotusarvoisen käyttäytymisen analyysi:
  - Vaihe 1 toimii aina ajassa $O(n+|Pocc|)$ (vakioaakkostolla), joten voidaan keskittyä vaiheen 2 analyysiin.
  - Merkitään luvulla $E(r,n)$ r:n pituisen hahmon (palasen) odotusarvoista esiintymien määrää tekstissä, jonka pituus on n; oletetaan, että hahmon ja tekstin merkit on riippumattomasti ja tasaisesti valittu aakkostosta, jonka koko on σ.
  - Saadaan arvio
    $$E(r,n) = \frac{n-r+1}{\sigma^r} \leq \frac{n}{\sigma^r}.$$
  - Olkoon hahmo P jaettu palasiin $P = P^1P^2\cdots P^{k+1}$, missä kukin $|P^i| \geq r$ ja r on suurin mahdollinen. Siis $r = \lfloor m/(k+1)\rfloor$.
  - Käyttämällä odotusarvon lineaarisuutta, odotusarvo $k+1$:n palasen yhteenlasketulle esiintymämäärälle, on korkeintaan $(k+1)E(r,n)$. Tämä on siis odotusarvo luvulle $|Pocc|$, eli vaiheen 1 antaman mahdollisten esiintymien joukon koolle.
  - Jos käytetään dynaamista ohjelmointia tarkistusvaiheessa, yhden esiintymäkandidaatin tarkistamisen aikavaatimus on $O(m^2)$. Siten
    $$g(m,n,k,|Pocc|) = \mathcal{O}(m^2(k+1)E(r,n)) = \mathcal{O}(m^3\frac{n}{\sigma^r}),$$
    missä on käytetty arviota $k \leq m$.
  - Tarkistusvaihe toimii odotusarvoisesti lineaarisessa ajassa jos $m^3\frac{n}{\sigma^r} \leq c\cdot n$, jollain vakiolla $c > 0$. Ratkaisemalla tämä r:n suhteen saadaan
    $$r \geq 3\log_\sigma m - \log_\sigma c = \Omega(\log_\sigma m)$$

– Siis tarkistusvaihe voidaan ratkaista odotusarvoisesti lineaarisessa ajassa kun

$$k+1 \leq m/r = \mathcal{O}\left(\frac{m}{\log_\sigma m}\right)$$

Lause 2.5.2 *Olkoon hahmo $P = p_1 \cdots p_m$ ja teksti $S = s_1 \cdots s_n$. Yllä kuvattu seulontamenetelmä toimii odotusarvoisesti ajassa $\mathcal{O}(n)$, kun $k = \mathcal{O}(m/\log_\sigma m)$.*

- Vaihe 2 voidaan itse asiassa ratkaista odotusarvoisesti ajassa $\mathcal{O}(n/m)$, kun yllä $c \cdot n$ korvataan $c \cdot n/m$:llä. Ehto $k = \mathcal{O}(m/\log_\sigma m)$ pysyy ennallaan.
- Vaihe 1 voidaan myös ratkaista odotusarvoisesti ajassa $\mathcal{O}(n\frac{\log_\sigma m}{r}) = \mathcal{O}(nk\frac{\log_\sigma m}{m})$ käyttäen optimaalista monen hahmon etsimiseen tarkoitettua algoritmia (esim. MultiBDM (Crochemore & Rytter: Text Algorithms. Oxford University Press, 1994)).
- Saadaan siis algoritmi, joka on odotusarvoisesti lineaarinen, kun $k = \Theta(m/\log_\sigma m)$, ja alilineaarinen, kun $k = o(m/\log_\sigma m)$.
- Ongelmaan on kehitetty useita muitakin odotusarvoisesti lineaarisia/alilineaarisia ratkaisuja.
- Chang & Marr (Proc. 5th Symp. on Combinatorial Pattern Matching, 259–273, 1994) todistavat, että $\Omega(n\frac{(k+\log_\sigma m)}{m})$ on alaraja ongelman odotusarvoiselle vaativuudelle. He antavat myös optimaalisen algoritmin tapaukseen $k < \frac{m}{3} - \mathcal{O}(\frac{m}{\sqrt{\sigma}})$.

Luku 3

Merkkijonojen järjestäminen ja hakurakenteet

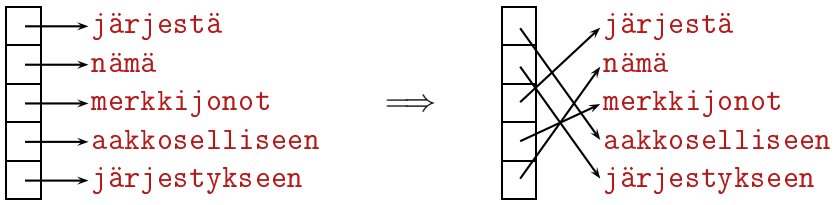
- Järjestämisalgoritmit ja hakurakenteet, kuten binääripuut, ovat kaikkein perustavinta lajia olevia algoritmeja ja tietorakenteita.

- Niillä on myös läheinen yhteys toisiinsa. Järjestetty taulukko mahdollistaa nopean binäärihaun. Toisaalta järjestäminen voidaan tehdä lisäämällä alkiot binääripuuhun ja poimimalla ne sieltä järjestyksessä.

- Alkioiden vertailuihin perustuvia perusalgoritmeja ja -tietorakenteita voi käyttää myös, kun alkiot ovat merkkijonoja, mutta vertailujen hitauden vuoksi algoritmit hidastuvat. Tässä luvussa nähdään, että erityisesti merkkijonoille tarkoitetuilla algoritmeilla ja tietorakenteilla päästään parempiin tuloksiin.

- Huomaa, että ongelmana on nimenomaan vertailut, ei esimerkiksi siirtely.

Esimerkki 3.0.3



3.1 Merkkijonovertailuihin perustuva järjestäminen

- Alkioiden vertailuihin perustuvia järjestämisalgoritmeja on lukuisia, esim. lisäys-, lomitus-, keko-, ja pikajärjestäminen.
- Tunnetusti jokainen vertailuihin perustuva algoritmi tarvitsee vähintään $\Omega(n \log n)$ vertailua n:n alkion järjestämiseen (sekä pahimmassa että keskimääräisessä tapauksessa). Monille algoritmeille $\mathcal{O}(n \log n)$ vertailua myös riittää.
- Kun alkiot ovat merkkijonoja, $\mathcal{O}(n \log n)$ vertailua ei kuitenkaan tarkoita aikavaatimusta $\mathcal{O}(n \log n)$.

Määritelmä 3.1.1 *Olkoon $A = A[0 \ldots m) = a_0 a_1 \cdots a_{m-1}$ ja $B = B[0 \ldots n) = b_0 b_1 \cdots b_{n-1}$ kaksi merkkijonoa järjestetyssä aakkostossa Σ, $|\Sigma| = \sigma$. Jono A on* aakkosjärjestyksessä *pienempi kuin B, merkitään $A < B$, jos ja vain jos joko*

- *$m < n$ ja $A = B[0 \ldots m)$ (eli A on B:n aito alkuosa) tai*
- *on olemassa $0 \leq i < \min\{m, n\}$, jolla $A[0 \ldots i) = B[0 \ldots i)$ ja $a_i < b_i$.*

Lause 3.1.2 *Olkoon $\mathcal{R} = \{S_1, S_2, \ldots, S_n\}$ joukko merkkijonoja järjestetyssä aakkostossa Σ, $|\Sigma| = \sigma$. Joukon $\mathcal{R}$ järjestäminen aakkosjärjestykseen yleisellä alkioiden vertailuihin perustuvalla algoritmilla vaatii $\Omega(n \log n \log_\sigma n)$ merkkivertailua.*

Todistus.

- Olkoon $k = \lfloor (\log_\sigma n)/2 \rfloor$. Olkoon $\mathcal{R}_\alpha$, $\alpha \in \sigma^k$, niiden $\mathcal{R}$:n jonojen joukko, joiden alkuosa on α. Siis $\mathcal{R}_\alpha = \{S \in \mathcal{R} \mid \alpha \text{ on } S\text{:n alkuosa}\}$. Olkoon $n_\alpha = |\mathcal{R}_\alpha|$ kaikilla $\alpha \in \sigma^k$.
- Tarkastellaan joukon $\mathcal{R}_\alpha$ jonojen keskinäisissä vertailuissa tarvittavien merkkivertailujen määrää:
  - Jokainen keskinäinen vertailu vaatii vähintään $|\alpha| = k$ merkkivertailua.
  - Mikään jonovertailu, jonka osapuolista ainakin toinen on $\mathcal{R}_\alpha$:n ulkopuolelta, ei tuota mitään informaatiota $\mathcal{R}_\alpha$:n sisäisestä järjestyksestä.

  - Järjestämisen yleisen alarajan perusteella tarvitaan $\Omega(n_\alpha \log n_\alpha)$ $\mathcal{R}_\alpha$:n sisäistä jonovertailua ja siis $\Omega(kn_\alpha \log n_\alpha)$ merkkivertailua.

- Yhteensä merkkivertailuja tarvitaan siis ainakin $\Omega(k \sum_{\alpha \in \sigma^k} n_\alpha \log n_\alpha)$.

- Koska $\sum_{\alpha \in \sigma^k} n_\alpha > n/2$ ja $x \log x$ on konkaavi funktio (derivaatta on kasvava eli toinen derivaatta on suurempi kuin 0 kaikilla $x > 0$), pätee

$$\sum_{\alpha \in \sigma^k} n_\alpha \log n_\alpha \geq n \log(n/\sigma^k) \geq n \log \sqrt{n} = \frac{1}{2} n \log n \ .$$

□

- Vakioaakkostolla yleinen algoritmi vaatii siis ainakin $\Omega(n \log^2 n)$ ajan n:n merkkijonon järjestämiseen.

- Merkkijonojen erikoisluonteen huomioonottavalle algoritmille tämä alaraja ei päde. Merkkivertailuihin perustuvalle algoritmille saadaan kuitenkin löyhempi alaraja.

Määritelmä 3.1.3 *Olkoon S merkkijono ja $\mathcal{R} = \{S_1, \ldots, S_n\}$ joukko merkkijonoja. Jonon S yksilöivä alkuosa joukossa $\mathcal{R}$ on lyhin S:n alkuosa, joka erottaa sen kaikista muista $\mathcal{R}$:n jonoista. Merkitään S:n yksilöivän alkuosan pituutta $dp_\mathcal{R}(S)$ ja merkitään $\mathcal{R}$:n jonojen yksilöivien alkuosien pituuksien summaa $DP(\mathcal{R}) = \sum_{i=1}^{n} dp_\mathcal{R}(S_i)$.*

Esimerkki 3.1.4 *Yksilöivät alkuosat:*

```
a akkoselliseen
järjesty kseen
järjestä
m erkkijonot
n ämä
```

Lause 3.1.5 *Olkoon $\mathcal{R} = \{S_1, S_2, \ldots, S_n\}$ joukko merkkijonoja järjestetyssä aakkostossa Σ, $|\Sigma| = \sigma$. Joukon $\mathcal{R}$ järjestäminen aakkosjärjestykseen merkkivertailuihin perustuvalla algoritmilla vaatii $\Omega(DP(\mathcal{R}) + n \log n)$ merkkivertailua.*

Todistus.

- Joukon järjestäminen vaatii selvästi, että kaikki yksilöivien alkuosien merkit tutkitaan. Tästä saadaan alaraja $\Omega(DP(\mathcal{R}))$.
- Yleisen järjestämisen alarajan perusteella pätee alaraja $\Omega(n \log n)$.

□

3.2 Merkkijonopikajärjestäminen

> Jon L. Bentley & Robert Sedgewick: Fast Algorithms for Sorting and Searching Strings. Teoksessa *Proc. 8th Annual ACM–SIAM Symposium on Discrete Algorithms*, sivut 323–350, 1997.

- Pikajärjestäminen on osoittautunut käytännössä nopeimmaksi yleiskäyttöiseksi järjestämisalgoritmiksi.
- Tarkastellaan seuraavaksi merkkijonojen järjestämiseen muokattua variaatiota pikajärjestämisestä.

Ternääripikajärjestäminen

- Esitetään ensin peruspikajärjestämisen muunnelma, joka perustuu jakoon kolmeen osaan tavallisen kahden sijasta.

Algoritmi 3.2.1 *Ternääripikajärjestä(R)*

Syöte: (Moni)joukko R.
Tuloste: Joukon R alkiot nousevassa järjestyksessä.
(1) **if** $|R| \leq 1$ **then return** R;
(2) valitse jakoalkio $x \in R$;
(3) $R_< := \{s \in R \mid s < x\}$;
(4) $R_= := \{s \in R \mid s = x\}$;
(5) $R_> := \{s \in R \mid s > x\}$;
(6) $R_< :=$ Ternääripikajärjestä$(R_<)$;
(7) $R_> :=$ Ternääripikajärjestä$(R_>)$;
(8) **return** $R_< \cdot R_= \cdot R_>$;

- Algoritmi on tässä esitetty abstraktissa muodossa, joka tuo sen rakenteen selkeästi esiin. Käytännön toteutuksessa joukko R on talletettu taulukkoon ja sen osajoukot ovat taulukon osavälejä.
- Normaali pikajärjestäminen jakaa joukon R kahteen osaan $R_{\leq}$ ja $R_{\geq}$, joista molemmat voivat sisältää joukon $R_{=}$ alkioita. Jako kolmeen on hivenen kahteenjakoa hitaampi käytännössä, mutta vain hivenen.
- Kuten normaalissa pikajärjestämisessä, algoritmin nopeus riippuu jakoalkion valinnasta:
  - Optimaalinen jakoalkio, R:n mediaani, on mahdollista laskea lineaarisessa ajassa, jolloin pahimman tapauksen aikavaatimus on $\mathcal{O}(n \log n)$.
  - Jakoalkion valinta satunnaisesti tuottaa odotusarvoiseksi aikavaatimukseksi $\mathcal{O}(n \log n)$.
  - Käytännössä kannattaa valita muutaman alkion otoksen mediaani.

  Oletetaan jatkossa optimaalinen jakoalkion valinta.

Merkkijonopikajärjestäminen

- Merkkijonopikajärjestäminen on kuin ternääripikajärjestäminen, mutta se tekee jaon vain yhden merkkiposition, ei koko merkkijonon perusteella.

Algoritmi 3.2.2 *Merkkijonopikajärjestä($\mathcal{R}$,ℓ)*

Syöte: Joukko $\mathcal{R}$ merkkijonoja ja niiden yhteisen alkuosan pituus ℓ.
Tuloste: Joukko $\mathcal{R}$ aakkosjärjestyksessä.

(1) **if** $|\mathcal{R}| \leq 1$ **then return** $\mathcal{R}$;
(2) valitse jakoalkio $X \in \mathcal{R}$;
(3) $\mathcal{R}_{<} := \{S \in \mathcal{R} \mid S[\ell] < X[\ell]\}$;
(4) $\mathcal{R}_{=} := \{S \in \mathcal{R} \mid S[\ell] = X[\ell]\}$;
(5) $\mathcal{R}_{>} := \{S \in \mathcal{R} \mid S[\ell] > X[\ell]\}$;
(6) $\mathcal{R}_{<} :=$ Merkkijonopikajärjestä($\mathcal{R}_{<}, \ell$);
(7) $\mathcal{R}_{=} :=$ Merkkijonopikajärjestä($\mathcal{R}_{=}, \ell + 1$);
(8) $\mathcal{R}_{>} :=$ Merkkijonopikajärjestä($\mathcal{R}_{>}, \ell$);
(9) **return** $\mathcal{R}_{<} \cdot \mathcal{R}_{=} \cdot \mathcal{R}_{>}$;

- Ensimmäisessä kutsussa $\ell = 0$.

Esimerkki 3.2.3 *Mahdollinen jako eräällä syötteellä, kun* $\ell = 2$.

al	p	habet		al	i	gnment
al	i	gnment		al	g	orithm
al	l	ocate		al	i	as
al	g	orithm	$\Longrightarrow$	al	l	ocate
al	t	ernative		al	l	
al	i	as		al	p	habet
al	t	ernate		al	t	ernative
al	l			al	t	ernate

Lause 3.2.4 *Merkkijonopikajärjestäminen järjestää merkkijonojoukon* $\mathcal{R} = \{S_1, S_2, \ldots S_n\}$ *ajassa* $\mathcal{O}(DP(\mathcal{R}) + n \log n)$.

Todistus.

- Riveillä 3–5 suoritetut vertailut hallitsevat aikavaatimusta.
- Käytetään tasoitettua analyysiä merkkivertailujen määrän analyysiin. Vertailun kustannus veloitetaan joko merkkijonolta tai yksittäiseltä merkiltä vertailun tuloksen perusteella:
  1. $S[\ell] = X[\ell]$: Veloitetaan kustannus merkiltä $S[\ell]$.
     - Nyt merkkijono S joutuu joukkoon $\mathcal{R}_=$, jonka järjestävässä rekursiokutsussa yhteisen alkuosan pituus kasvaa $\ell + 1$:een. Siten merkki $S[\ell]$ ei osallistu jatkossa vertailuihin ja sille tuleva kokonaiskustannus on yksi.
     - Algoritmi ei tutki merkkejä, jotka ovat yksilöivien alkuosien ulkopuolella. Siten merkkivertailuja, joiden tulos on yhtäsuuruus, on enintään $DP(\mathcal{R})$.
  2. $S[\ell] \neq X[\ell]$: Veloitetaan kustannus merkkijonolta S.
     - Nyt merkkijono S joutuu joukkoon $\mathcal{R}_<$ tai $\mathcal{R}_>$. Olettaen optimaalinen jakoalkion valinta, tämän joukon koko on enintään $|\mathcal{R}|/2$.
     - Jokainen jonolta S veloitettava vertailu siis puolittaa S:n sisältävän joukon koon. Siten S:lle tuleva kokonaislasku on enintään $\log n$.
- Siis yhtäsuuruusvertailuja on enintään $DP(\mathcal{R})$ ja erisuuruusvertailuja enintään $n \log n$. □