Luku 3

Merkkijonojen järjestäminen ja hakurakenteet

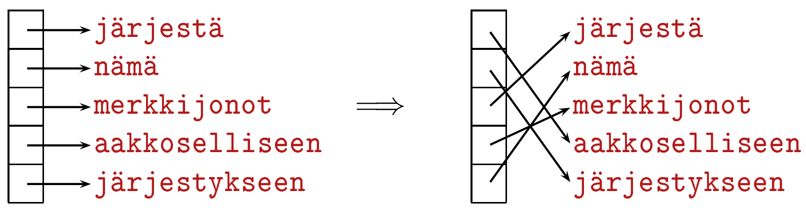
- Järjestämisalgoritmit ja hakurakenteet, kuten binääripuut, ovat kaikkein perustavinta laatua olevia algoritmeja ja tietorakenteita.

- 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 3.0.2



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 $A = A[0 \dots m) = a_0 a_1 \cdots a_{m-1}$ ja $B = B[0 \dots n) = b_0 b_1 \cdots b_{n-1}$ 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 $A = B[0 \dots m)$ tai*
- *on olemassa $0 \leq i < \min\{m, n\}$, jolla $A[0 \dots i) = B[0 \dots i)$ ja $a_i < b_i$.*

Lause 3.1.2 *Olkoon $\mathcal{R} = \{S_1, S_2, \dots, S_n\}$ 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 $\sum_{\alpha\in\sigma^k} n_\alpha = n$ 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 $\mathcal{O}(DP(\mathcal{R}) + n \log n)$ ja tilassa $\mathcal{O}(1)$.*

Todistus.

- Kuten normaali pikajärjestäminen, merkkijonopikajärjestäminen on mahdollista toteuttaa vakiotilassa ja siten, että jakoa tehtäessä (rivit 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 $n \log n$.

$\square$

- Merkkijonopikajärjestäminen on siis optimaalinen merkkivertailuihin perustuva merkkijonojen järjestämisalgoritmi.
- Se on myös käytännössä nopea.

3.3 Kantalukujärjestäminen

- Kokonaislukuja on mahdollista järjestää alarajaa $\Omega(n \log n)$ nopeammin, jopa lineaarisessa ajassa.
- Jos aakkosto koostuu kokonaisluvuista, myös merkkijonoja voi järjestää alarajaa $\Omega(DP(\mathcal{R}) + n \log n)$ nopeammin.
- $\Omega(DP(\mathcal{R}))$ on kuitenkin aina alaraja, kun merkkejä on käsiteltävä yksittäin.

Laskemisjärjestäminen

- Laskemisjärjestäminen järjestää pienet kokonaisluvut lineaarisessa ajassa.

Algoritmi 3.3.1 *Laskemisjärjestäminen*

Syöte: (Moni)joukko $R = \{k_0, k_1, \ldots k_{n-1}\}$ kokonaislukuja väliltä $[0, \sigma)$.
Tuloste: Joukko R nousevassa järjestyksessä taulukossa $J[0 \ldots n)$.

for $i := 0$ **to** $\sigma - 1$ **do** $C[i] := 0$;
for $i := 0$ **to** $n - 1$ **do** $C[k_i] := C[k_i] + 1$;
$sum := 0$;
for $i := 1$ **to** $\sigma - 1$ **do**
    $tmp := C[i]$; $C[i] := sum$; $sum := sum + tmp$;
for $i := 0$ **to** $n - 1$ **do**
    $J[C[k_i]] = k_i$;
    $C[k_i] := C[k_i] + 1$;
return J;

- Aika- ja tilavaatimus ovat $\mathcal{O}(n + \sigma)$.
- Algoritmi 3.3.1 on *vakaa* eli se säilyttää samanarvoisten alkioiden keskinäisen järjestyksen.

Lopusta alkuun kantalukujärjestäminen

- Kantalukujärjestämisessä (radix sort) laskemisjärjestämistä sovelletaan yhteen positioon kerrallaan.
- Olkoon Laskemisjärjestä($\mathcal{R}$,ℓ) kutsu, joka järjestää merkkijonojoukon $\mathcal{R}$ position ℓ perusteella käyttäen laskemisjärjestämistä.
- Viimeisestä positiosta alkuun päin etenevä järjestäminen on erittäin yksinkertainen.

Algoritmi 3.3.2 *Lopusta alkuun kantalukujärjestäminen*

Syöte: Joukko $\mathcal{R} = \{S_0, S_1, \ldots S_{n-1}\}$ m:n pituisia merkkijonoja aakkostossa $[0, \sigma)$.
Tuloste: Joukko $\mathcal{R}$ aakkosjärjestyksessä
(1) **for** $\ell := m - 1$ **to** 0 **do** Laskemisjärjestä($\mathcal{R}$,ℓ);
(2) **return** $\mathcal{R}$;

Esimerkki 3.3.3

$$\begin{array}{l} \texttt{cat}\\ \texttt{him}\\ \texttt{ham}\\ \texttt{bat} \end{array} \Longrightarrow \begin{array}{l|l|} \cline{2-2} \texttt{hi} & \texttt{m}\\ \texttt{ha} & \texttt{m}\\ \texttt{ca} & \texttt{t}\\ \texttt{ba} & \texttt{t}\\ \cline{2-2} \end{array} \Longrightarrow \begin{array}{l|l|l} \cline{2-2} \texttt{h} & \texttt{a} & \texttt{m}\\ \texttt{c} & \texttt{a} & \texttt{t}\\ \texttt{b} & \texttt{a} & \texttt{t}\\ \texttt{h} & \texttt{i} & \texttt{m}\\ \cline{2-2} \end{array} \Longrightarrow \begin{array}{|l|l} \hline \texttt{b} & \texttt{at}\\ \texttt{c} & \texttt{at}\\ \texttt{h} & \texttt{am}\\ \texttt{h} & \texttt{im}\\ \cline{1-1} \end{array}$$

- Algoritmissa oletettiin, että kaikki jonot ovat samanpituisia. Se on kuitenkin helppo yleistää eripituisille merkkijonoille (HT).

Lause 3.3.4 *Olkoon $\mathcal{R} = \{S_0, S_1, \ldots S_{n-1}\}$ joukko merkkijonoja aakkostossa $[0, \sigma)$. Lopusta alkuun kantalukujärjestäminen järjestää $\mathcal{R}$:n ajassa $\mathcal{O}(||\mathcal{R}|| + m\sigma)$ ja tilassa $\mathcal{O}(n + \sigma)$, missä $||\mathcal{R}||$ on merkkijonojen kokonaispituus ja m on pisimmän jonon pituus.*

- Algoritmin heikkous on, että se tutkii merkkijonot kokonaan eli vie $\Omega(||\mathcal{R}||)$ ajan silloinkin, kun $DP(\mathcal{R})$ on paljon pienempi kuin $||\mathcal{R}||$.
- Se soveltuukin lähinnä lyhyille samanmittaisille merkkijonoille.
- Tärkeä sovellusalue on myös kokonaislukujen järjestäminen esittämällä ne σ-kantajärjestelmän jonoina.

Alusta loppuun kantalukujärjestäminen

- Kantalukujärjestämisen voi myös aloittaa alkupäästä. Tällöin tarvitaan kuitenkin hieman monimutkaisempaa kirjanpitoa.

- Tuloksena oleva algoritmi on itseasiassa paljon merkkijonopikajärjestämisen kaltainen. Kolmeenjaon sijasta tehdäänkin nyt jako σ:aan osaan.

al	p	habet		al	g	orithm
al	i	gnment		al	i	gnment
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

- Oletetaan nyt, että Laskemisjärjestä($\mathcal{R}$,ℓ) palauttaa joukot $\mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_{\sigma-1}$, missä $\mathcal{R}_i = \{S \in \mathcal{R} \mid S[\ell] = i\}$.

- Rekursiiviset kutsut johtavat lopulta hyvin pieniin joukkoihin. Laskemisjärjestäminen kuluttaa kuitenkin aina ajan $\Omega(\sigma)$. Algoritmin nopeuttamiseksi alle σ:n kokoisiin joukkoihin käytetään merkkijonopikajärjestämistä.

Algoritmi 3.3.5 *Kantalukujärjestä($\mathcal{R}$,ℓ)*

Syöte: Joukko $\mathcal{R}$ merkkijonoja ja niiden yhteisen alkuosan pituus ℓ.
Tuloste: Joukko $\mathcal{R}$ aakkosjärjestyksessä.
if $|\mathcal{R}| < \sigma$ **then return** Merkkijonopikajärjestä($\mathcal{R}$,ℓ);
$(\mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_{\sigma-1})$ = Laskemisjärjestä($\mathcal{R}$,ℓ);
for $i := 0$ **to** $\sigma - 1$ **do** Kantalukujärjestä($\mathcal{R}_i$,$\ell + 1$);
return $\mathcal{R}_0 \cdot \mathcal{R}_1 \cdots \mathcal{R}_{\sigma-1}$;

Lause 3.3.6 *Olkoon $\mathcal{R} = \{S_0, S_1, \ldots S_{n-1}\}$ joukko merkkijonoja aakkostossa $[0, \sigma)$. Alusta loppuun kantalukujärjestäminen järjestää $\mathcal{R}$:n ajassa $\mathcal{O}(DP(\mathcal{R}) + n \log \sigma)$ ja tilassa $\mathcal{O}(n + \sigma)$.*

Todistus.

- Tarkastellaan aikavaatimusta, kun algoritmia kutsutaan osajoukolla $\mathcal{R}'$, $|\mathcal{R}'| = k$.

- Kun $k \geq \sigma$, ei-rekursiiviset osat, mukaanlukien laskemisjärjestäminen, toimivat ajassa $\mathcal{O}(k)$. Samalla tutkitaan k merkkiä ja samoja merkkejä ei tutkita enää uudelleen.
- Yksilöivien alkuosien ulkopuolisia merkkejä ei tutkita. Siis merkkijonopikajärjestäminen poisluettuna aikavaatimus on yhteensä $\mathcal{O}(DP(\mathcal{R}))$.
- Merkkijonopikajärjestämistä kutsutaan vain σ:a pienemmille joukoille ja kukin jono on mukana enintään yhdessä kutsussa. Siten merkkijonopikajärjestämisessä kuluva kokonaisaika on $\mathcal{O}(DP(\mathcal{R}) + n \log \sigma)$.

$\square$

- Jos σ on vakio, algoritmi toimii optimaalisessa ajassa $\mathcal{O}(DP(\mathcal{R}))$.
- Käytännössä, tavuaakkostolla, se on suunnilleen yhtä nopea kuin merkkijonopikajärjestäminen.

3.4 Kaksivaiheinen kantalukujärjestäminen

R. Paige ja R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing* 16(6):973-989, 1987.

- Edellä kuvattu alusta loppuun kantalukujärjestäminen ei ole aivan optimaalinen suuremmilla aakkostoilla, mikä johtuu pienten joukkojen järjestämiseen kuluvasta ajasta.
- Erityisesti, jos $\sigma = \Omega(n)$, se ei ole merkkijonopikajärjestämistä nopeampi.
- Esitetään nyt menetelmä, jolla päästään aikavaatimukseen $\mathcal{O}(DP(\mathcal{R})+ \sigma)$.

Pienen joukon laskemisryhmittely ja -järjestäminen

- Laskemisjärjestäminen (algoritmi 3.3.1) järjestää n kokonaislukua väliltä $[0, \sigma)$ ajassa $\mathcal{O}(n + \sigma)$. Pienille joukoille ($n < \sigma$), haluttaisiin kuitenkin päästä aikavaatimukseen $\mathcal{O}(n)$.

- Aikavaatimukseen $\mathcal{O}(n)$ päästään, jos riittää *ryhmitellä* alkiot eli järjestää ne siten, että samanarvoiset alkiot ovat peräkkäin, mutta muuten järjestyksellä ei ole väliä.
- Lisäksi oletetaan, että käytössä on valmiiksi alustettu σ:n kokoinen laskuritaulukko.
- Ideana on pitää listaa L niistä laskuritaulukon alkioista, jotka ovat nollaa suurempia.

Algoritmi 3.4.1 *Laskemisryhmittely*

Syöte: (Moni)joukko $R = \{k_0, k_1, \ldots k_{n-1}\}$ kokonaislukuja väliltä $[0, \sigma)$.
Tuloste: Joukko R ryhmiteltynä taulukossa $J[0 \ldots n)$.
Aputila: Taulukko $C[0 \ldots \sigma)$, joka alussa ja lopussa sisältää vain nollia.

(1) $L := \emptyset$;
(2) **for** $i := 0$ **to** $n - 1$ **do**
(3)   **if** $C[k_i] = 0$ **then** $L := L \cup \{k_i\}$;
(4) $C[k_i] := C[k_i] + 1$;
(5) $sum := 0$;
(6) **for each** $k \in L$ **do**
(7) $tmp := C[k]$; $C[k] := sum$; $sum := sum + tmp$;
(8) **for** $i := 0$ **to** $n - 1$ **do**
(9) $J[C[k_i]] = k_i$;
(10) $C[k_i] := C[k_i] + 1$;
(11) **for each** $k \in L$ **do** $C[k] := 0$;
(12) **return** J;

- Algoritmi voidaan muuttaa järjestämisalgoritmiksi lisäämällä riville 5 listan L järjestäminen.
- Jos lista L saadaan syötteenä valmiiksi järjestettynä, myös järjestäminen voidaan tehdä ajassa $\mathcal{O}(n)$.

Kaksivaiheinen kantalukujärjestäminen

Kaksivaiheisen kantalukujärjestämisen ideana on suorittaa alusta loppuun kantalukujärjestäminen kahteen kertaan, ensin ryhmitellen sitten järjestäen.

- Vaihe 1: Kantalukuryhmittely
  - Suorita alusta loppuun kantalukujärjestäminen kuten algoritmissa 3.3.5 mutta käytä laskemisryhmittelyä laskemisjärjestämisen sijaan.

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

  - Pienillekään joukoille ei nyt käytetä merkkijonopikajärjestämistä.
  - Kaikki laskemisryhmittelyssä syntyneet listat L otetaan talteen.

- Välivaihe: Järjestä listat L
  - Yhdistä kaikki listat, mutta pidä kirjaa kunkin alkion alkuperäisestä listasta.
  - Järjestä yhdistetty lista käyttäen normaalia laskemisjärjestämistä.
  - Palauta alkiot alkuperäisiin listoihin, nyt järjestettynä.

- Vaihe 2: Kantalukujärjestäminen
  - Toimitaan kuten vaiheessa 1, mutta laskemisryhmittelyn sijasta suoritetaan laskemisjärjestäminen käyttämällä valmiiksi järjestettyjä listoja L.
  - Jotta kussakin vaiheessa tarvittava lista L löydetään helposti, suoritetaan rekursiokutsut samassa järjestyksessä kuin vaiheessa 1. Oikea kutsujärjestys voidaan helposti laskea välivaiheessa.

Lause 3.4.2 *Olkoon $\mathcal{R} = \{S_0, S_1, \ldots S_{n-1}\}$ joukko merkkijonoja aakkostossa $[0, \sigma)$. Kaksivaiheinen kantalukujärjestäminen järjestää $\mathcal{R}$:n ajassa $\mathcal{O}(DP(\mathcal{R}) + \sigma)$ ja tilassa $\mathcal{O}(n + \sigma)$.*

Hyvin suuret aakkostot

- Jos σ on hyvin suuri, voidaan kukin merkki jakaa osiin, niin että saadaan pidempiä merkkijonoja pienemmässä aakkostossa. Esimerkiksi:

```
83 111 114 116
      ||
      V
01010011 01101111 01110010 01110100
      ||
      V
0101 0011 0110 1111 0111 0010 0111 0100
      ||
      V
5 3 6 15 7 2 7 4
```

- Jos merkit jaetaan k osaan saadaan algoritmi, jonka aikavaatimus on $\mathcal{O}(k \cdot DP(\mathcal{R}) + \sigma^{1/k})$ ja tilavaatimus $\mathcal{O}(n + \sigma^{1/k})$.
- Jos $\sigma > n$, voidaan valita $k = \lceil \log_n \sigma \rceil$, jolloin $\sigma^{1/k} \leq n$. Tällöin aikavaatimus on $\mathcal{O}(DP(\mathcal{R}) \log_n \sigma)$ ja tilavaatimus $\mathcal{O}(n)$.
- Jos $\sigma = n^{\mathcal{O}(1)}$, eli jos yksi merkki mahtuu $\mathcal{O}(\log n)$ bittiin, k on vakio ja aikavaatimus on $\mathcal{O}(DP(\mathcal{R}))$ eli algoritmi on optimaalinen.

3.5 Hakurakenteet

- Siirrytään nyt järjestämisestä hakurakenteisiin.
- Yleisesti, hakurakenne on tietorakenne, joka tallentaa joukon R ja mahdollistaa seuraavien operaatioiden nopean suorituksen:
  - Lisäys
  - Poisto
  - Haku: Tutki, sisältääkö R alkion x, ja jos sisältää, palauta siihen mahdollisesti liittyvät tiedot.
  - Naapurihaku: Jos R ei sisällä alkiota x, palauta sen lähin edeltäjä tai seuraaja R:ssä.

- Naapurihaku mahdollistaa esim. osavälihaut: Etsi kaikki R:n alkiot, jotka ovat välillä $[x, y]$.
- Kaikki hakurakenteet eivät tue naapurihakua, esimerkkinä hajautustaulu.
- Ylläoleva koskee *dynaamisia* hakurakenteita. *Staattinen* hakurakenne ei mahdollista lisäystä ja poistoa, ainoastaan hakurakenteen muodostamisen annetulle joukolle R. Esimerkiksi järjestetty taulukko on staattinen hakurakenne.

Alkuosahaut

- Merkkijonojoukolle $\mathcal{R}$ voidaan yllämainittujen lisäksi määritellä hakuoperaatiot:
  - Alkuosahaku: Tutki, onko S jonkin $\mathcal{R}$:n jonon alkuosa. Jos on, palauta kaikki sellaiset jonot.
  - Pisin yhteinen alkuosa: Jos S ei ole minkään $\mathcal{R}$:n jonon alkuosa, mikä on S:n pisin alkuosa, joka on.
- Alkuosahaut ovat läheisessä yhteydessä naapurihakuun.
  - Jos $\mathcal{R}$:ssä on jonoja, joiden alkuosa on S, S:n lähin seuraaja $\mathcal{R}$:ssä on ensimmäinen niistä ja loput ovat sen välittömiä seuraajia.
  - Jonolla S on pisin yhteinen alkuosa sen edeltäjän tai seuraajan kanssa.
- Tarkastellaan jatkossa pääasiassa alkuosahakua.

Perushakurakenteet merkkijonoille

- Perushakurakenteita, kuten binääripuita, voidaan käyttää myös merkkijonoille.
- Kuten edellä nähtiin, myös alkuosahaut voidaan toteuttaa naapurihaun avulla. (Toisaalta esim. hajautustaulu ei tue alkuosahakuja.)
- Operaatiot kuitenkin hidastuvat vertailujen hitauden vuoksi.