ALGORITMITEKNIIKKA
Syksy 2000

Pekka Orponen ja Jarmo Ernvall
Jyväskylän yliopisto

Esipuhe

Tämä moniste perustuu Helsingin, Toronton ja Jyväskylän yliopistoissa 1980- ja 1990-lukujen mittaan useaan kertaan pitämiimme tietotekniikan laudatur-peruskurssitasoisiin algoritmitekniikan luentoihin. Luentojen käsittelemä asiakokonaisuus on kansainvälisesti jokseenkin vakiintunut, esikuvanaan Ahon, Hopcroftin ja Ullmanin klassikkoteos "The Design and Analysis of Computer Algorithms" vuodelta 1974. Algoritmien suunnittelun ja analyysin menetelmiä on toki vuosien mittaan tullut lisää, esimerkkeinä vaikkapa tässäkin monisteessa käsitellyt itsejärjestyvät tietorakenteet ja algoritmien tasattu analyysi, mutta tietty tekniikoiden ydinjoukko on osoittanut pysyvyytensä. Keskeinen osa tämänkin monisteen algoritmeista, analyyseista ja esimerkeistä periytyy tuosta Ahon, Hopcroftin ja Ullmanin kirjasta sekä samaa aihepiiriä käsittelevästä suomenkielisestä käsikirjoituksesta "Algoritmien suunnittelu ja analyysi" (Orponen, Tarhio & Ukkonen 1992).

Jyväskylän yliopistossa tämän monisteen mukaiset algoritmitekniikan opinnot sisältyvät kolmen opintoviikon laajuiseen tietotekniikan valinnaiseen laudaturkurssiin, jonka sen valinneet opiskelijat tyypillisesti suorittavat kolmantena opintovuonnaan. Tätä ennen he ovat jo toisen vuoden cum laude -tasoisilla kursseilla tutustuneet tavanomaisiin perustietorakenteisiin (pinot, jonot, puut, verkot) kurssilla "Tietorakenteet ja algoritmit 1" (3 ov) sekä yleisiin algoritmien suunnitteluperiaatteisiin (osittaminen, taulukointi, "ahneus", peruutus jne.) kurssilla "Tietorakenteet ja algoritmit 2" (2 ov).

Nyt käsillä olevan monisteen luvut 1–5 ovat pääosin ensimmäisen tekijän (PO) ja luku 6 toisen tekijän (JE) käsialaa. Suurimman osan monisteen raskaasta LaTeX-puhtaaksikirjoitustyöstä on kesien 1999 ja 2000 aikana suorittanut fil. yo Tuukka Tawast, mistä hänelle suurkiitos. Painokuntoon tekstin on viimeistellyt JE.

Jyväskylässä, 15.11.2000

Tekijät

Sisältö

Luku 1

Algoritmien analyysimenetelmiä

1.1 Iteratiivisten algoritmien analysointi

Esimerkki: lisäyslajittelu.

Algoritmi (pseudo - Pascal)

1.**procedure** insertsort $(A[1..n])$;
2. **for** $i := 2$ to n **do**
3. **begin** $x :=$ A$[i]$; $j := i - 1$;
4. **while** $j > 0$ **and** $x <$ A$[j]$ **do**
5. **begin** A$[j + 1] :=$ A$[j]$;
6. $j := j - 1$
7. **end**;
8. A$[j + 1] := x$
9.**end**.

Pahimman tapauksen aikavaativuusanalyysi:

Analysoidaan silmukat sisältä ulospäin:

$T_{5-6}\ (n, i, j)\ \leq\ c_1$ [vain alkeisoperaatioita]

$T_{4-7}\ (n, i)\ \leq\ c_2 + (i - 1)c_1$ [alustus + enintään $(i - 1)$ sisemmän silmukan kierrosta]

$T_{3-9}\ (n,i)\ \leq\ c_3+c_2+(i-1)c_1$ [$c_3\ \sim$ rivin 3 alkeisoperaatioiden kustannus]

$$\begin{aligned} T_{2-9}\ (n)\ &\leq\ c_4+\sum_{i=2}^{n}(c_3+c_2+(i-1)c_1) \\ &=\ c_4+(n-1)(c_3+c_2)+c_1\sum_{i=2}^{n}(i-1) \\ &\leq\ cn+\frac{1}{2}n(n-1)\ =\ O(n^2). \end{aligned}$$

1.2 Aikavaativuuden analysointisääntöjä

Sovellettava harkiten.

Merkitään: *T(P)* = algoritmin P aikavaativuus.

- *T(*$x := e$*)* = vakio, T(read x) = vakio, T(write x) = vakio

  Poikkeuksia: Rakenteisten muuttujien käsittely voi vaatia enemmän aikaa. Samoin, jos lauseke e sisältää proseduurikutsuja.

- T($S_1; S_2; ...; S_k$) $=\ T(S_1)+...+T(S_k)\ =\ O(\max\{T(S_1),...,T(S_k)\})$

- T(**if** P **then** S_1 **else** S_2)

$$= \begin{cases} T(P)+T(S_1) & \text{jos P = true} \\ T(P)+T(S_2) & \text{jos P = false} \end{cases}$$

- T(**while** P **do** S) = vakio + suorituskertojen lkm $\cdot$ (T(P)+T(S))

Sisäkkäisiä silmukoita analysoidessa edetään sisältä ulospäin. Tällöin ulompien silmukoiden ohjausmuuttujat (yms.) ovat sisempien silmukoiden analyysin parametreja.

1.3 Rekursiivisten algoritmien analysointi

Esimerkki: Lomituslajittelu.

Algoritmi:	Aikavaativuus
procedure mergesort (A[1..n]);	$T(n)$ yht.
if n=1 **then return**	$O(1)$

else begin

tarvitaan aputaulukot $U[1 \ldots \lfloor n/2 \rfloor]$, $V[1 \ldots \lceil n/2 \rceil]$:	
$U := A[1 \ldots \lfloor n/2 \rfloor]$;	$O(n/2)$=$O(n)$
$V := A[\lfloor n/2 \rfloor + 1 \ldots n]$;	$O(n/2)$=$O(n)$
mergesort(U);	$T(\lfloor n/2 \rfloor)$
mergesort(V);	$T(\lceil n/2 \rceil)$
merge(U, V, A);	$O(n)$

end.

procedure merge (*U[1..k], V[1..l], A[1..n]*);

{lomittaa lajitellut taulukot U, V taulukkoon A ajassa $O(n)$}

Oletetaan yksinkertaisuuden vuoksi, että n on kahden potenssi.

Algoritmin aikavaativuuden analysointi johtaa tällöin seuraavaan *rekursio*t. *differenssiyhtälöön*:

$$T(n) \leq \begin{cases} c_1 & , \text{kun } n = 1 \\ \underbrace{2T\left(\frac{n}{2}\right)}_{\text{rek. kutsut}} + \underbrace{c_2 n}_{\text{muu laskenta}} & , \text{kun } n \geq 2 (\text{so. } n = 2^k,\ k > 0) \end{cases}$$

Yhtälö voidaan ratkaista naiivisti *purkamalla*:

Oletetaan $n \geq 2$; tällöin

$$\begin{aligned} T(n) &\leq 2T\left(\tfrac{n}{2}\right) + c_2 n \\ &\leq 2(2T\left(\tfrac{n}{4}\right) + c_2 \cdot \tfrac{n}{2}) + c_2 n \ = \ 4T\left(\tfrac{n}{4}\right) + 2c_2 n \\ &\leq \ldots \\ &\leq 2^i T\left(\tfrac{n}{2^i}\right) + i \cdot c_2 n \\ &\leq \ldots \\ &\leq 2^{log_2 n} \cdot T(1) + log_2 n \cdot c_2 n \\ &= c_1 n + c_2\ n\ log_2 n \ = \ O(n\ log_2 n) \end{aligned}$$

Eräs käyttökelpoinen yleistys

Tarkastellaan rekursioyhtälöitä muotoa

$$(*) \begin{cases} T(1) &= c_1 \\ T(n) &= aT(\frac{n}{b}) + d(n), \text{ kun } n = b^k,\ b = 1, 2, .., \end{cases}$$

Puretaan yhtälö, kun $n = b^k,\ k \geq 1$:

$$\begin{aligned}
T(n) &= T(b^k) \\
&= aT(b^{k-1}) + d(b^k) \\
&= a(aT(b^{k-2}) + d(b^{k-1})) + d(b^k) \\
&= a^2T(b^{k-2}) + ad(b^{k-1}) + d(b^k) \\
&= a^2(aT(b^{k-3}) + ad(b^{k-2})) + ad(b^{k-1}) + d(b^k) \\
&= a^3T(b^{k-3}) + a^2d(b^{k-2}) + ad(b^{k-1}) + d(b^k) \\
&= \cdots
\end{aligned}$$

$$= a^iT(b^{k-i}) + \sum_{j=0}^{i-1} a^jd(b^{k-j}) \;\; = \cdots = a^kT(1) + \sum_{j=0}^{k-1} a^jd(b^{k-j})$$

$$= \underbrace{c_1a^k}_{\text{homog. osa} = c_1n^{log_ba}} + \underbrace{\sum_{j=0}^{k-1} a^jd(b^{k-j})}_{\text{epähomog. osa}}$$

Tarkastellaan erityisesti tärkeää erityistapausta $d(r) = cr^\alpha$. Tällöin saadaan:

$$\begin{aligned}
T(n) &= c_1a^k + \sum_{j=0}^{k-1} a^jd(b^{k-j}) \\
&= c_1a^k + \sum_{j=0}^{k-1} a^j \cdot cb^{\alpha k - \alpha j} \\
&= c_1a^k + cb^{\alpha k} \sum_{j=0}^{k-1} (\frac{a}{b^\alpha})^j
\end{aligned}$$

Oletetaan $a \neq b^\alpha$:

$$\begin{aligned}
&= c_1a^k + cb^{\alpha k} \cdot \frac{(\frac{a}{b^\alpha})^k - 1}{\frac{a}{b^\alpha} - 1} \\
&= c_1a^k + c \cdot \frac{a^k - b^{\alpha k}}{\frac{a}{b^\alpha} - 1} \qquad\qquad |k = log_bn \\
&= c_1a^{log_bn} + c \cdot \frac{a^{log_bn} - b^{\alpha log_bn}}{\frac{a}{b^\alpha} - 1} \\
&= c_1n^{log_ba} + c \cdot (\frac{a}{b^\alpha} - 1)^{-1} \cdot (n^{log_ba} - n^\alpha)
\end{aligned}$$

Lause. Rekursioyhtälön

$$\begin{cases}
T(1) & = c_1 > 0 \\
T(n) & = aT(\frac{n}{b}) + cn^\alpha, \text{kun } n = b^k, k = 1, 2, ..
\end{cases}$$

ratkaisun kertaluokka on:

1. $T(n) = \theta(n^{log_b a}), jos\ a > b^\alpha\ tai\ c = 0;$
2. $T(n) = \theta(n^\alpha), jos\ a < b^\alpha\ ja\ c \neq 0;$
3. $T(n) = \theta(n^\alpha log\ n), jos\ a = b^\alpha\ ja\ c \neq 0;$

Todistus.

Kohdat 1 ja 2 seuraavat suoraan laskun tuloksesta. Kohdassa 3 saadaan:

$$T(n) = c_1 a^k + cb^{\alpha k} \cdot k = c_1 b^{\alpha k} + cb^{\alpha k} \cdot k \ = c_1 n^\alpha + cn^\alpha log_b n. \ \square$$

1.4 Algoritmien keskimääräisen vaativuuden analysointi

Palautetaan mieliin keskimääräisen vaativuuden määritelmä (tarkastellaan tässä vain aikavaativuutta):

$$T_{ave}(n) = \sum_{|x|=n} p_n(x) \cdot T(x),$$

missä $p_n(x)$ on n:n kokoisten tapausten todennäköisyys ja $T(x)$ on algoritmin vaativuus syötteellä x.

Tyypillisesti algoritmin vaativuus ei riipu syötteen x kaikista yksityiskohdista, vaan sen jonkin ominaisuuden (joidenkin ominaisuuksien) arvosta (arvoista) $k(x), T(x) = T(k(x))$. Tällöin keskimääräisen tapauksen analysointi voidaan osittaa:

$$T_{ave}(n) = \sum_k P(k(x) = k) \cdot T(k),$$

so. analysoidaan erikseen algoritmin vaativuus parametrin arvolla k ja todennäköisyys, että satunnainen syöte antaa arvon k.

Esimerkki: Lisäyslajittelu

```
procedure insertsort (A[1..n]);
for i := 2 to n do
begin x := A[i]; j := i-1;
        while j > 0 and x < A[j] do
        begin A[j+1] := A[j];
                j := j-1
```

```
        end;
        A[j+1] := x
end.
```

Analyysi:

Syötteiden jakauma: Oletetaan, että syötetaulukon $A[1..n]$ alkiot ovat kaikki erisuuruisia, ja että alkioiden kaikki järjestykset (permutaatiot) ovat yhtä todennäköisiä.

Merkitään:

P(i,k) = todennäköisyys, että kun ulommalla silmukka-muuttujalla on arvo i, sisempi silmukka (rivit 5-6) suoritetaan k kertaa.

Algoritmin keskimääräinen vaativuus on tällöin (joillakin c_1, c_2, c_3)

$$\begin{aligned} T_{ave}(n) &= c_1 + \sum_{i=2}^{n}(c_2 + \sum_{k=0}^{i-1} P(i,k) \cdot kc_3) \\ &= c_1 + (n-1)c_2 + c_3 \sum_{i=2}^{n}\sum_{k=0}^{i-1} P(i,k) \cdot k \end{aligned}$$

Havaitaan, että taulukon alkiota $A[i] = x$ tarkasteltaessa ovat alkiot $A[1]$,..., $A[i-1]$ jo järjestyksessä, ja

P(i, k) = todennäköisyys, että joukossa $A[1..i-1]$ on k kpl x:ää suurempia alkioita.

Jakaumaoletuksesta seuraa, että P(i, k) on sama kaikille

$$P(i,k) = \frac{1}{i}, k = 0, .., i-1.$$

Keskimääräinen vaativuus on siten

$$\begin{aligned}
T_{ave}(n) &= c_1 + (n-1)c_2 + c_3 \sum_{i=2}^{n} \sum_{k=0}^{i-1} \frac{k}{i} \\
&= c_1 + (n-1)c_2 + c_3 \sum_{i=2}^{n} \frac{1}{i} \sum_{k=0}^{i-1} k \\
&= c_1 + (n-1)c_2 + c_3 \sum_{i=2}^{n} \frac{1}{i} \cdot \frac{1}{2} i(i-1) \\
&= c_1 + (n-1)c_2 + \frac{c_3}{2} \sum_{i=1}^{n-1} i \\
&= c_1 + (n-1)c_2 + \frac{c_3}{4} n(n-1) \\
&= \Theta(n^2).
\end{aligned}$$

Lisäyslajittelu on siis (tarkastellun jakaumaoletuksen vallitessa) keskimääräiseltäkin käyttäytymiseltään neliöllinen algoritmi.

Esimerkki: Pikalajittelu (Quicksort) (alkiot erisuuruisia)

Syöte: Taulukko A[*1..n*] (tapauksen koko = *n*)

Tulos: Taulukko A[*1..n*] lajiteltuna nousevaan järjestykseen.

Idea: Proseduurikutsu quicksort(i, j) lajittelee osataulukon A[*i..j*] seuraavasti:

1. Valitse jokin jakoalkio *v* taulukosta A[*i..j*]
2. Järjestä A[*i..j*] s.e. jollakin *k*:
   - *v*:tä pienemmät alkiot tulevat osataulukkoon A[*i..k-1*]
   - *v*:tä suuremmat tai yhtä suuret alkiot tulevat osataulukkoon A[*k..j*]
3. Kutsu quicksort(*i, k-1*) ja quicksort(*k, j*)

Jakoalkio pitäisi valita siten, että taulukko puolittuu mahdollisimman tasaisesti. Valinta täytyy kuitenkin tehdä nopeasti. Mahdollisuuksia:

1. Satunnainen valinta.
2. Selataan taulukon alusta kunnes on löydetty kaksi eri arvoa. Suurempi valitaan.

Seuraavassa tarkastellaan valintaperiaatetta 2.

Algoritmi(t):

```
function partition (i, j, v);
{Järjestää taulukon A[i..j] siten, että
        alkiot <v tulevat osaan A[i..k-1] ja
        alkiot ≥v tulevat osaan A[k..j].
Palauttaa arvon k.}
begin
        l:=i;
        r:=j;
        while l ≤ r do
            begin
                while A[l] < v do l:=l+1;
                while A[r] ≥ v do r:=r-1;
                if l < r then
                begin t:=A[l];A[l]:=A[r];A[r]:=t end
            end;
        return l
end;


procedure quicksort (i, j);
if i < j then
begin
          v:= pivot(i, j);
          k:= partition(i, j, v);
          quicksort(i, k-1);
          quicksort(k, j)
end.
```

Analyysi:

(a) *Väite*: Operaation partition(i, j, v) aikavaativuus on O(j-i).
Perustelu: Olkoon k kutsun partition(i, j, v) palauttama arvo. Jokaisen silmukan "**while** $l \leq r$ **do**..."suorituskerralla l kasvaa tai r pienenee tai molemmat. Toisaalta l kasvaa $i \rightarrow k$ ja r pienenee $j \rightarrow k-1$, joten operaatio "l:=l+1"suoritetaan k-i kertaa ja operaatio "r:=r+1"suoritetaan j-k+1 kertaa, yhteensä j-i+1 kertaa. Alkioiden vaihto A[l] $\leftrightarrow$ A[r] suoritetaan $\leq \min\{k\text{-}i,\ j\text{-}k+1\}$ kertaa. Kaikkiaan siis O(j-i) operaatiota.□

(b) *Väite*: Jakoalkion valinta pivot(i, j) voidaan toteuttaa ajassa O(j-i).
Perustelu: HT.□

Tehdään sitten syötteestä seuraava jakaumaoletus:

(*)Taulukon A[*1..n*] alkiot ovat kaikki erisuuria, ja kaikki A:n järjestykset ovat yhtä todennäköisiä.

(*Huom.* Tällöin on aina pivot(*i, j*) = max{A[*i*], A[*i+1*]}.)

(c) *Väite*: Jakaumaoletuksen (*) vallitessa on algoritmin quicksort(*1, n*) keskimääräinen aikavaativuus luokkaa O(*n log n*).

Huom. Algoritmin pahimman tapauksen aikavaativuus on $\Omega(n^2)$ (Tod. HT.)

Perustelu: Merkitään

P(k) = todennäköisyys, että valittu jakoalkio v = pivot(*1, n*) on taulukon k:nneksi pienin alkio ($k \geq 2$):

Tällöin saadaan aikavaativuudelle T(n) = $T_{ave}(n)$ palautuskaava:

$$T(n) \leq c_1, \quad n = 1$$
$$T(n) \leq \sum_{k=2}^{n} P(k) \cdot (T(k-1) + T(n-k+1)) + c_2 n, \quad n \geq 2$$

Jakaumaoletuksen ja valitun jakoalkion määritystavan takia on

$$\begin{aligned} P(k) &= Pr(A[1] \quad k\text{:nneksi pienin ja } A[2] < A[1]) \\ &\quad + Pr(A[2] \quad k\text{:nneksi pienin ja } A[1] < A[2]) \\ &= \frac{1}{n} \cdot \frac{k-1}{n-1} + \frac{1}{n} \cdot \frac{k-1}{n-1} = \frac{2(k-1)}{n(n-1)} \end{aligned}$$

Tapauksessa n $\geq$ 2 saadaan siis:

$$
\begin{aligned}
T(n) &\leq \sum_{k=2}^{n} \frac{2(k-1)}{n(n-1)}(T(k-1)+T(n-k+1))+c_2 n \\
&= \frac{2}{n(n-1)} \sum_{k=1}^{n-1} k(T(k)+T(n-k))+c_2 n \\
&= \frac{2}{n(n-1)}(\sum_{k=1}^{n-1} kT(k)+\sum_{k=1}^{n-1} kT(n-k)+c_2 n \\
&= \frac{2}{n(n-1)}(\sum_{k=1}^{n-1} kT(k)+\sum_{k=1}^{n-1}(n-k)T(k))+c_2 n \\
&= \frac{2}{n(n-1)} \sum_{k=1}^{n-1} nT(k)+c_2 n \\
&= \frac{2}{(n-1)} \sum_{k=1}^{n-1} T(k)+c_2 n
\end{aligned}
$$

Jos siis määritellään funktio $\hat{T}(n)$ rekursioyhtälöllä

$$
\begin{aligned}
\hat{\mathrm{T}}(1) &= 1 \\
\hat{T}(n) &= \frac{2}{n-1} \sum_{k=1}^{n-1} \hat{T}(k)+n, \quad n \geq 2
\end{aligned}
$$

niin kaikilla n on $\mathrm{T}(n) \leq c\ \hat{T}(n)$, missä $c = \max\{c_1, c_2\}$.

Ratkaistaan rekursioyhtälö "historian eliminoinnilla":

$$
\begin{aligned}
(n-1)\hat{T}(n) &= 2\sum_{k=1}^{n-1} \hat{T}(k)+n(n-1) \\
(n-2)\hat{T}(n-1) &= 2\sum_{k=1}^{n-2} \hat{T}(k)+(n-1)(n-2) \\
\therefore (n-1)\hat{T}(n)-(n-2)\hat{T}(n-1) &= 2\hat{T}(n-1)+2(n-1) \\
\therefore \hat{T}(n) &= \frac{n}{n-1}\hat{T}(n-1)+2.
\end{aligned}
$$

Lasku voidaan nyt viedä loppuun yksinkertaisesti purkamalla

$$\begin{aligned}
\hat{T}(n) &= \frac{n}{n-1}\hat{T}(n-1) + 2 \\
&= \frac{n}{n-1}(\frac{n-1}{n-2}\hat{T}(n-2) + 2) + 2 \\
&= \frac{n}{n-2}\hat{T}(n-2) + 2(\frac{n}{n-1} + 1) \\
&= \frac{n}{n-2}(\frac{n-2}{n-3}\hat{T}(n-3) + 2) + 2(\frac{n}{n-1} + \frac{n}{n}) \\
&= \frac{n}{n-3}\hat{T}(n-3) + 2n(\frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}) \\
&= \ldots \\
&= n\hat{T}(1) + 2n(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}) \\
&= n + 2n(H_n - 1),
\end{aligned}$$

missä H_n on harmonisen sarjan $1+\frac{1}{2}+\frac{1}{3}+\ldots$ n:s osasumma. Tunnetusti on

$$H_n = ln\ n + \Theta(1)$$

joten

$$\hat{T}(n) = 2n\ ln\ n + \Theta(n),$$

ja siten myös $T(n) = \Theta(n\ log\ n)$.□

1.5 Tasattu vaativuus (amortized complexity)

Idea: Tarkastellaan yksittäisten operaatioiden kustannusten sijaan pitkien toimenpidejonojen kustannuksia. Pitkissä jonoissa saattavat (esim. tietorakenteen muuttumisen takia) kalliit operaatiot olla harvinaisia. Tasataan kustannukset siten, että halvoilta operaatioilta otetaan aikaa "säästöön"kalliita varten.

Esimerkki: Binäärilaskuri.

Tarkastellaan taulukossa A[*0..k*] toteutetun binäärilaskurin kasvattamista (A[0] vähiten merkitsevä bitti):

procedure inc(A[*0..k*])
begin *i:=0*;

```
        while i < k and A[i]=1 do
        begin A[i]:=0; i:=i+1 end;
        if A[i]=0 then A[i]:=1
                  else OVERFLOW
end.
```

Mikä on n peräkkäisen inc-operaation kustannus?

Oletetaan yksinkertaisuuden vuoksi, että aluksi $A \equiv 0$ ja $n \leq 2^k$.

Naiivi arvio: kunkin operaation kustannus on verrannollinen sen muuttamien bittien määrään, joka on pahimmassa tapauksessa $\Theta(k)$. Siis n operaation kustannus on *O(nk)*.

Tasatun vaativuuden käyttö: kohdat 1.5.1 ja 1.5.2.

1.5.1 Kokonaiskustannusmenetelmä

Tarkastellaan koko operaatiojonon kustannusta, otetaan erityisesti huomioon operaatioiden vaikutukset tietorakenteisiin.

Usein on edullista laskea kustannukset operaatioiden suoritusten sijaan tietorakenteiden muutoksista. ($\sim$ summausjärjestyksen vaihto)

Esimerkki binäärilaskuri: "laskutetaan"kutakin bittiä A[i] sen arvon muuttuessa. Koko jonon kustannus on tällöin = bittien laskujen loppusumma.

Kuvaannollisesti: Jos merkitään

COST(j) = j:nnen inc-operaation kustannus, j=1,..,n
CHARGE(i) = bitin A[i] kokonaislasku, i= 0,..,k

niin

$$TOTAL(n) = \sum_{j=1}^{n} COST(j) = \sum_{i=0}^{k} CHARGE(i).$$

Esimerkki: k=3, n=8:

			$\leftarrow i$		
A:	3	2	1	0	COST(j)
	0	0	0	0	-
j	0	0	0	$\underline{1}$	1
$\downarrow$	0	0	$\underline{1}$	$\underline{0}$	2
	0	0	1	$\underline{1}$	1
	0	$\underline{1}$	$\underline{0}$	$\underline{0}$	3
	0	1	0	$\underline{1}$	1
	0	1	$\underline{1}$	$\underline{0}$	2
	0	1	1	$\underline{1}$	1
	$\underline{1}$	$\underline{0}$	$\underline{0}$	$\underline{0}$	4
CHARGE(i)	1	2	4	8	$\sum = 15$

Havaitaan, että n peräkkäisessä inc-operaatiossa:

bitin A[0] arvo vaihtuu n kertaa
bitin A[1] arvo vaihtuu $\lfloor n/2 \rfloor$ kertaa
$\vdots$
bitin A[k] arvo vaihtuu $\lfloor n/2^k \rfloor$ kertaa

(Oletetaan, ettei tapahdu ylivuotoa.)

Siten:

$$TOTAL(n) = \sum_{i=0}^{k} \lfloor \frac{n}{2^i} \rfloor \leq n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n.$$

1.5.2 Potentiaalimenetelmä

Liitetään käsiteltävään tietorakenteeseen D "sopiva"potentiaalifunktio $\Phi(D)$, jota halvat operaatiot kasvattavat ja kalliit pienentävät.

Määritellään operaatiojonossa $op_1, ..., op_n$ operaaton op_i tasattu kustannus:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}),$$

missä c_i on operaation todellinen kustannus ja $\Phi(D_i)$ on tietorakenteen potentiaali operaation i jälkeen.

Analyysissä arvioidaan todellisten kustannusten sijaan tasattuja kustannuksia; näistä saadaan lopulta arvio operaatiojonon todellisille kokonaiskustannuksille, koska:

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0),\end{aligned}$$

so.

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i - \Phi(D_n) + \Phi(D_0).$$

Esimerkki binäärilaskuri:

$\Phi(A) \triangleq$ taulukon A sisältämien ykkösbittien määrä.

Tarkastellaan inc-operaatiota, joka muuttaa t ykköstä nollaksi ja yhden nollan ykköseksi:

$$\begin{aligned}\hat{c} &= c + \Phi' - \Phi \\ &= t + 1 + (1 - t) \\ &= 2.\end{aligned}$$

Siten n operaation jonon kokonaiskustannus on:

$$\begin{aligned}\sum_{i=1}^{n} c_i &= \sum_{i=1}^{n} \hat{c}_i - \Phi(A_n) + \Phi(A_0) \\ &= 2n - (\text{ykkösten määrä lopussa}) + (\text{ykkösten määrä alussa})\end{aligned}$$

1.6 Rekursioyhtälöiden ratkaisumenetelmiä

Edellä on jo tutustuttu rekursioyhtälöiden ratkaisemiseen purkamalla. Seuraavassa käsitellään lisää tekniikoita:

1. Arvauksen sovittaminen
2. Arvo- ja määrittelyalueiden muunnokset
3. Generoivat funktiot

1.6.1 Arvauksen sovittaminen

Tarkastellaan rekursio(epä)yhtälöä muotoa

$$\begin{cases} T(1) & \leq c \\ T(n) & \leq g(T(1), ..., T(n-1), n), \ n \geq 2, \end{cases}$$

missä g on ensimmäisten n-1 argumentin suhteen ei-vähenevä fumktio.

Tämä voidaan yrittää ratkaista seuraavasti:

(i) Arvataan ratkaisun muoto: $T(n) \leq f(n, \bar{a})$, missä $\bar{a}$ on parametrivektori.

(ii) Yritetään sovittaa parametrit $\bar{a}$ niin, että voidaan todistaa

$$f(1, \bar{a}) \geq c$$
$$f(n, \bar{a}) \geq g(f(1, \bar{a}), ..., f(n-1, \bar{a}), n).$$

Induktiolla seuraa tällöin, että

$$T(n) \leq f(n, \bar{a}) \text{ kaikilla } n \geq 1.$$

Esimerkki: Lomituslajittelun analyysi.

Yhtälö:

$$\begin{cases} T(1) & \leq c_1 \\ T(n) & \leq 2T(n/2) + c_2 n, \ n \geq 2 \end{cases}$$

Arvataan:

$$T(n) \leq an \ log_2 \ n + b \text{ kaikilla } n.$$

Sovitetaan:

$$\begin{aligned} a \cdot 1 \ log_2 \ 1 + b = b \geq c_1 \quad &\Leftarrow \text{ Tosi, jos } b \geq c_1 \\ an \ log_2 \ n + b &\geq 2(a\frac{n}{2} log_2 \ \frac{n}{2} + b) + c_2 n \\ &= an \ log_2 \ n - an + 2b + c_2 n \\ \Leftrightarrow an - b \geq c_2 n \quad &\Leftarrow \text{ Tosi, jos } a - \frac{b}{n} \geq c_2 \end{aligned}$$

Voidaan valita esimerkiksi $b = c_1, a = c_1 + c_2$:

$$T(n) \leq (c_1 + c_2) n \ log_2 \ n + c_1 \text{ kaikilla } n.$$

1.6.2 Muunnostekniikat

Esimerkki: Arvoalueen muunnos.

Tarkastellaan yhtälöä

$$(*)\begin{cases} T(0) & = 1 \\ T(n) & = 3T(n-1)^2,\ n \geq 1 \end{cases}$$

Jotta hankalan näköisestä neliöstä päästään eroon, tehdään muunnos $U(n) = logT(n)$ ja korvataan (*) yhtälöllä

$$\begin{cases} U(0) & = log_2\ T(0) = log_2\ 1 = 0 \\ U(n) & = log_2\ T(n) = log_2(3T(n-1)^2) = 2logT(n-1) + log_2\ 3 = 2U(n-1) + log_2\ 3 \end{cases}$$

Purkamalla saadaan tälle ratkaisu:

$$U(n) = (2^n - 1)log_2\ 3$$

Alkuperäisen yhtälön ratkaisu on siis:

$$T(n) = 2^{U(n)} = 2^{(2^n-1)log_2\ 3} = 3^{2^n-1}.$$

Esimerkki: Määrittelyalueen muunnos (so. muuttujanvaihto).

Tarkastellaan yhtälöä

$$\begin{cases} T(2) & = 1 \\ T(n) & = 2T(\sqrt{n}) + 1,\ \text{“sopivilla” } n > 2. \end{cases}$$

Neliöjuuresta päästään eroon tekemällä muuttujanvaihto $n = 2^k$ ja tarkastelemalla funktion T(n) sijaan funktiota $U(k) = T(2^k)$:

$$\begin{cases} U(1) & = T(2) = 1 \\ U(k) & = T(2^k) = 2T(2^{k/2}) + 1 = 2U(k/2) + 1. \end{cases}$$

Purkamalla saadaan tälle ratkaisu $U(k) = 2k - 1$. Alkuperäisen yhtälön ratkaisu on siis:

$$T(n) = 2log_2\ n - 1,\ \text{“sopivilla” } n.$$

1.6.3 Generoivat funktiot

Ylivertaisesti voimakkain rekursioyhtälöiden analysointitapa; yksinkertaisissa tapauksissa kuitenkin raskas.

Tarkastellaan jotakin lukujonoa $t =< t_0, t_1, t_2, ... >$. (Esim. $t_n = T(n)$ jostakin rekursioyhtälöstä).

Jonon t generoiva funktio on reaalimuuttujan (tai yleisimmin kompleksimuuttujan) z t-kertoimisen potenssisarjan määrittelemä funktio:

$$G(z) = \sum_{n \geq 0} t_n z^n.$$

Esimerkkejä:

(i) Jonon $< 1, 1, 1, ... >$ generoiva funktio on

$$G(z) = \sum_{n \geq 0} 1 \cdot z^n = \sum_{n \geq 0} z^n = \frac{1}{1-z}.$$

(ii) Jonon <1,2,4,8,...> generoiva funktio on

$$G(z) = \sum_{n \geq 0} 2^n z^n = \sum_{n \geq 0} (2z)^n = \frac{1}{1-2z}.$$

(iii) Jonon <0,1,2,3,...> generoiva funktio on

$$G(z) = \sum_{n \geq 0} n z^n = \frac{z}{(1-z)^2}$$

Annetun jonon generoiva funktio sisältää erittäin tiiviissä ja tehokkasti analysoitavassa muodossa tiedon kaikista jonon alkioista. Potenssisarjojen perusominaisuuksista seuraa, että jos G(z) on jonon $< t_0, t_1, ... >$ generoiva funktio, so.

$$G(z) = \sum_{n \geq 0} t_n z^n,$$

ja sarjan suppenemissäde origon ympärillä on > 0, niin kertoimet t_n saadaan yksikäsitteisesti kaavasta

$$t_n = \frac{1}{n!} G^{(n)}(0) \qquad |G^{(n)} = \text{ G:n } n\text{:s derivaatta}$$

Generoivilla funktioilla tehtävät laskelmat ovat usein päteviä, vaikka suppenemissäde olisi nollakin. Jatkossa ei kiinnitetä konvergenssikysymyksiin mitään huomiota; epävarmoissa tapauksissa arvoja t_n koskevat lopputulokset voi tarkastaa induktiolla.

Annetun rekursioyhtälön ratkaisussa generoivia funktioita käyttäen on kolme vaihetta.

(i) muodostetaan ratkaisujonon g.f.; [helppoa]

(ii) ratkaistaan funktio; [vaikeampaa]

(iii) määritetään kertoimet. [joskus hyvin vaikeaa]

Esimerkki 1. Tarkastellaan rekursioyhtälöä

$$\begin{cases} t_0 &= 1 \\ t_n &= 2t_{n-1}, \ n \geq 1. \end{cases}$$

Muodostetaan jonon $< t_0, t_1, t_2, ... >$ g.f.

$$\begin{aligned} G(z) &= \sum_{n\geq 0} t_n z^n \\ &= 1 \cdot z^0 + \sum_{n\geq 1} t_n z^n \\ &= 1 + \sum_{n\geq 1} 2t_{n-1} z^n \\ &= 1 + \sum_{n\geq 0} 2t_n z^{n+1} \\ &= 1 + 2z \sum_{n\geq 0} t_n z^n \\ &= 1 + 2zG(z). \end{aligned}$$

Ratkaistaan tästä funktio:

$$G(z) = \frac{1}{1-2z}.$$

Määritetään kertoimet: aiemman perusteella tiedetään, että

$$\frac{1}{1-2z} = \sum_{n\geq 0} 2^n z^n.$$

Koska (suppenevan) potenssisarjan kertoimet ovat yksikäsitteisesti määrätyt, on siis

$$t_n = 2^n.$$

Esimerkki 2. Tarkastellaan rekursioyhtälöä

$$\begin{cases} t_0 &= 1 \\ t_n &= 2t_{n-1} + 1 \end{cases}$$

Muodostetaan g.f.:

$$\begin{aligned} G(z) &= \sum_{n\geq 0} t_n z^n \\ &= 1 + \sum_{n\geq 1} (2t_{n-1} + 1) z^n \\ &= 1 + 2\sum_{n\geq 0} t_n z^{n+1} + \sum_{n\geq 1} z^n \\ &= 2z \sum_{n\geq 0} t_n z^n + \sum_{n\geq 0} z^n \\ &= 2zG(z) + \frac{1}{1-z}. \end{aligned}$$

Ratkaistaan tästä G:

$$G(z) = \frac{1}{(1-2z)(1-z)}$$

Kertoimen määrittämiseksi täytyy nyt muodostaa ensin rationaalifunktion $\frac{1}{(1-2z)(1-z)}$ osamurtolukuhajotelma:

$$\frac{1}{(1-2z)(1-z)} = \frac{A}{1-2z} + \frac{B}{1-z}, \qquad \text{A ja B vakioita.}$$

Vakiot A ja B saadaan selville helpoimmin laventamalla yhtälö vuorotellen tekijöillä $(1-2z), (1-z)$ ja ottamalla raja-arvot $z \to \frac{1}{2}, z \to 1$:

$$\begin{aligned} \frac{1}{1-z} &= A + B \cdot \frac{1-2z}{1-z} \implies_{z\to\frac{1}{2}} \quad A = 2 \\ \frac{1}{1-2z} &= A \cdot \frac{1-z}{1-2z} + B \implies_{z\to 1} \quad B = -1 \end{aligned}$$

Saadaan siis:

$$G(z) = \frac{2}{1-2z} - \frac{1}{1-z} = 2\sum_{n\geq 0} 2^n z^n - \sum_{n\geq 0} z^n = \sum_{n\geq 0} \underbrace{(2^{n+1}-1)}_{t_n} z^n.$$

Esimerkki 3. Fibonaccin luvut f_n määritellään yhtälöllä

$$\begin{cases} f_0 &= 0, f_1 = 1 \\ f_n &= f_{n-1} + f_{n-2},\ n \geq 2 \end{cases}$$

Muodostetaan tästä g.f.

$$\begin{aligned} F(z) &= \sum_{n\geq 0} f_n z^n \\ &= 0 + z + \sum_{n\geq 2} (f_{n-1} + f_{n-2}) z^n \\ &= z + \sum_{n\geq 1} f_n z^{n+1} + \sum_{n\geq 0} f_n z^{n+2} \\ &= z + zF(z) + z^2 F(z). \end{aligned}$$

Ratkaistaan tästä F:

$$F(z) = \frac{z}{1-z-z^2} = \frac{z}{(1-\varphi z)(1-\hat{\varphi} z)} \qquad \text{joillakin } \varphi, \hat{\varphi}$$

Ratkaistaan $\varphi, \hat{\varphi}$:

$$\begin{cases} \varphi\hat{\varphi} &= -1 \\ \varphi + \hat{\varphi} &= 1 \end{cases} \Rightarrow\ \varphi - \frac{1}{\varphi} = 1 \ \Rightarrow\ \varphi^2 - \varphi - 1 = 0$$

$$\Rightarrow \begin{cases} \varphi &= \frac{1}{2}(1+\sqrt{5}) \\ \hat{\varphi} &= \frac{1}{2}(1-\sqrt{5}) \end{cases} \text{ [tai toisinpäin]}$$

Osamurtolukuhajoitelmasta

$$F(z) = \frac{z}{(1-\varphi z)(1-\hat{\varphi} z)} = \frac{A}{1-\varphi z} + \frac{B}{1-\hat{\varphi} z}$$

saadaan

$$A = \lim_{z \to \frac{1}{\varphi}} \frac{z}{1 - \hat{\varphi} z} = \frac{\frac{1}{\varphi}}{1 - \frac{\hat{\varphi}}{\varphi}} = \frac{1}{\varphi - \hat{\varphi}} = \frac{1}{\sqrt{5}}$$
$$B = \lim_{z \to \frac{1}{\hat{\varphi}}} \frac{z}{1 - \varphi z} = \frac{\frac{1}{\hat{\varphi}}}{1 - \frac{\varphi}{\hat{\varphi}}} = \frac{1}{\hat{\varphi} - \varphi} = -\frac{1}{\sqrt{5}}.$$

G.f. voidaan siten kirjoittaa

$$\begin{aligned} F(z) &= \frac{1}{\sqrt{5}} \left(\frac{1}{1 - \varphi z} - \frac{1}{1 - \hat{\varphi} z} \right) \\ &= \frac{1}{\sqrt{5}} \left(\sum_{n \geq 0} \varphi^n z^n - \sum_{n \geq 0} \hat{\varphi}^n z^n \right) \\ &= \sum_{n \geq 0} \frac{1}{\sqrt{5}} \left(\varphi^n - \hat{\varphi}^n \right) z^n, \end{aligned}$$

mistä saadaan Fibonaccin luvuille suljettu muoto

$$f_n = \frac{1}{\sqrt{5}} (\varphi^n - \hat{\varphi}^n), \ n \geq 0.$$

Luku 2

Tietorakenteista

2.1 Joukko-operaatiot ja tietotyypit

Algoritmit voidaan usein suunnitella, tai ainakin esittää tasoittain:

(i) abstrakti versio joukko-operaatioiden avulla [kontrolli].

(ii) tarvittavan joukko-operaatioiden kokoelman tehokas toteutus [tietorakenne].

Usein tarvittavia joukko-operaatioita ovat esim:

1. member(a, S) : onko $a \in S$?
2. insert(a, S) : $S \leftarrow S \cup \{a\}$.
3. delete(a, S) : $S \leftarrow S$ -{a}.
4. replace(a, a', S) : $S \leftarrow (S$ -{a}$) \cup \{a'\}$.
5. union(S_i, S_j) : $S_i \leftarrow S_i \cup S_j$.
6. find(a) : jos $a \in \biguplus S_i$, niin se i, jolla $a \in S_i$; muuten ei määritelty.

   Jos alkioiden kesken on määritelty järjestys, niin mahdollisia ovat myös:

7. min(S) : min $\{a | a \in S\}$, jos $S \neq \emptyset$; muuten ei määritelty.
8. max(S) : max$\{a | a \in S\}$, jos $S \neq \emptyset$; muuten ei määritelty.
9. split(S_i, a, S_j) : $S_i \leftarrow \{b \in S_i | b \leq a\}, S_j \leftarrow \{b \in S_i | b > a\}$.

10. join$(S_i, S_j) : S_i \leftarrow S_i \cup S_j$, jos $\max(S_i) \leq \min(S_j)$; muuten ei määritelty.

Esimerkiksi yksinkertaisissa alkioiden kirjanpitotehtävissä tarvitaan tyypillisesti operaatioita member, insert, delete.

Toisaalta esimerkiksi Kruskalin algoritmissa tarvitaan kaarimetsän ylläpitoon operaatioita min, delete, find, union, insert.

Usein yhdessä esiintyvien operaatioiden kokoelmille ("abstrakteille tietotyypeille") on annettu omia nimiä, ja niitä varten on suunniteltu tehokkaita toteutuksia. Esimerkiksi:

Operaatiot	Nimi	Tehokkaita toteutuksia
member, insert, delete	hakemisto (engl. dictionary)	hajautustaulukot, tasapainoiset puut
insert, delete, min	prioriteettijono (engl. priority queue)	tasap.puut, keot, binomi- ja Fibonaccimetsät
insert, delete, min, union	yhdistyvä pr.jono (engl. mergeable heap)	binomi- ja Fibonacci-metsä
insert, delete, split, join, (min)	katenoitava jono (engl. concatenable q.)	(a, b)-puut

2.2 Yksinkertaisia toteutusrakenteita

Listat:

Joustava, mutta hidas toteutus:
-Member n-alkioisessa joukossa: $\Theta(n)$
-n insert-operaatiota: $\theta(n^2)$
-n - alkioisten joukkojen Union: $\Theta(n)$
(Huomattava parannus keskimääräisessä tapauksessa ns. hyppylistoilla (engl. skip lists))

Bittivektorit:

Tehokas pienillä perusjoukoilla, muuten tilaavievä ja hidas:
-Perusjoukko U, $|\mathrm{U}| \triangleq \mathrm{N}$
-Osajoukko $S \subseteq U$ esitetään n-bittisenä vektorina V_s:

$$V_s(i) = 1 \Longleftrightarrow i \in S.$$

Jos N ei liian suuri, useimmat joukko operaatiot voidaan toteuttaa bittivektorioperaatioina ajassa O(1).

Yleisesti kuitenkin operaatiot vaativat ajan $\approx \Theta(N)$ ja kukin joukko tilan $\Theta(N)$.

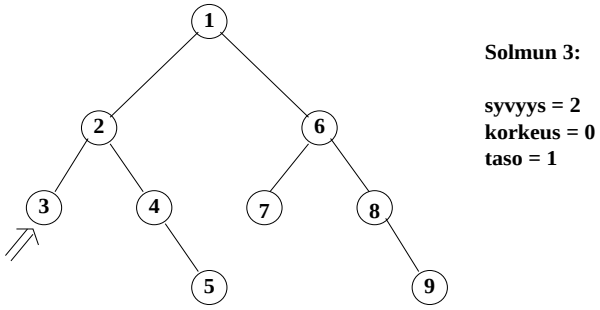
Hajautustaulut:

Lause: Hajautustaulujen avulla voidaan n member- insert- ja ja delete-operaatiota käsitellä keskimäärin ajassa $\Theta(n)$. Pahimmassa tapauksessa aikavaativuus on $\Theta(n^2)$.

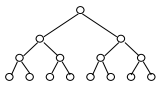
Osittaisia parannuksia: universaali ja täydellinen hajautus.

2.3 Yleistä binääripuista

Solmun syvyys = juuresta solmuun johtavan polun pituus.
Solmun korkeus = pisimmän solmusta lehteen johtavan polun pituus.
Puun korkeus = juuren korkeus.
Solmun taso = puun korkeus - solmun syvyys.



Täydellinen binääripuu:



Lemma:

(i) Binääripuussa on $\leq 2^p$ solmua, joiden syvyys on p.

(ii) Binääripuussa, jonka korkeus on k, on $\leq 2^{k+1} - 1$ solmua. Jos puu on täydellinen, solmuja on tasan $2^{k+1} - 1$.

(iii) n-solmuisen binääripuun korkeus on $\geq \lfloor log_2 n \rfloor$.□

Täydellinen binääripuu voidaan tallettaa kätevästi taulukkoon:

-juuri taulukon alkioksi 1.
-solmun i vasen poika alkioksi $2i$, oikea poika alkioksi $2i+1$

$$\Rightarrow \text{ solmun } k \text{ isä on paikassa } \lfloor k/2 \rfloor.$$

"Määritelmä". n - solmuinen puu on *tasapainoinen*, jos sen korkeus on $\approx log\ n$.

2.4 Binääriset hakupuut

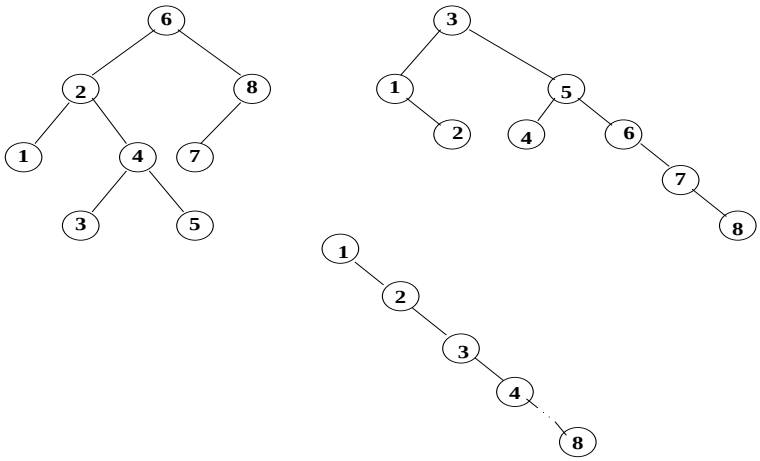
Määritelmä: järjestettyä joukkoa S esittävä binäärinen hakupuu on binääripuu, jonka kukin solmu v on nimetty alkiolla $l(v) \in S$ s.e.

(i) kullakin $a \in S$ on täsmälleen yksi v s.e. $l(v) = a$;

(ii) $l(u) < l(v)$ kaikilla u, jotka sijaitsevat v:n vasemmassa alipuussa:

(iii) $l(u) > l(v)$ kaikilla u, jotka sijaitsevat n:n oikeassa alipuussa.

Esimerkki: Joukkoa {1,2,...,8} esittäviä binäärisiä hakupuita:



Binääristen hakupuiden operaatiot:

-Member, Insert, Delete, Min

-Toteutukset TRA1(?)

Lause. Binäärisen hakupuun avulla voidaan n Member-, Insert- ja Min-operaatiota toteuttaa keskimäärin ajassa $O(n\ log\ n)$, jos kaikki operaatiojonot ovat yhtä todennäköisiä.□

Delete-operaatioiden analyysi on hankalampaa. Jos näitä on $\sim$ yhtä paljon kuin Insert-operaatioita, puut käyvät vinoiksi, eikä lause ehkä pidä paikkaansa.

Pahimmassa tapauksessa aikavaativuus on $\Omega(n^2)$. Näin on esimerkiksi, kun Insert-operaatiolla lisätään pitkiä järjestyksessä olevia jonoja. (Sovelluksissa tavallista!)

Tasapainottavilla puurakenteilla pahin tapaus saadaan luokkaan $O(n\ log\ n)$.

AVL-puut

(a, b) - puut $\rightarrow$ punamustat puut ja B-puut

splay -puut (=itsesäätävät binääriset hakupuut)

2.4.1 (a, b) - puut

Määritelmä: (a, b)-puu on (juurrettu, järjestetty) puu, jonka jokaisella sisäsolmulla on vähintään a ja enintään b jälkeläistä , ja jokainen lehti on samalla syvyydellä. (Puut eivät ole aina binäärisiä)

Tärkeitä erikoistapauksia:

- B-puut: (t,2t)-puita, missä t suuri. Tärkeä levytiedostojen talletusrakenne.

- (2,4) - puut ja erityisesti näiden toteus ns. punamustina puina. Tehokas tasapainoinen puurakenne keskusmuistikäsittelyyn. (Mutta operaatiot hieman mutkikkaita.)

Lemma. (a, b)-puussa, jonka korkeus on h, on $\frac{1}{a-1}(a^{h+1}-1) \leq$ solmujen määrä $\leq \frac{1}{b-1}(b^{h+1}-1)$ ja $a^h \leq$ lehtien määrä $\leq b^h$.$\Box$

Seuraus: (a, b) -puun, jossa on n lehteä, korkeus on $\Theta(log\ n)$.$\Box$

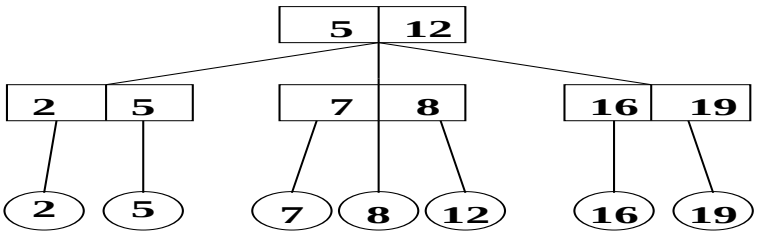
Tarkastellaan seuraavassa yksinkertaisuuden vuoksi vain (2, 3)-puita.

Joukon alkiot talletetaan puun lehtiin vasemmalta oikealle kasvavassa järjestyksessä.

Sisäsolmuissa kentät l ja m

=suurin solmun vasemmanpuoleisessa/keskimmäisessä alipuussa sijaitseva arvo [näin puuta voi käyttää hakupuuna]

Esimerkki: Joukko {2, 5, 7, 8, 12, 16, 19}.

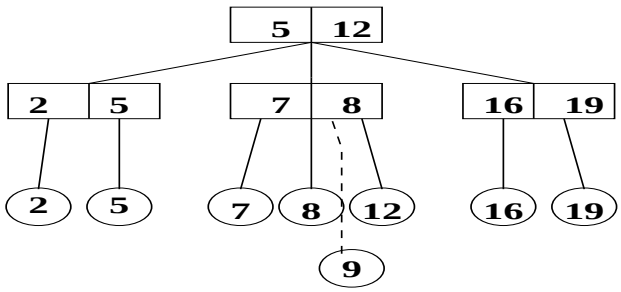


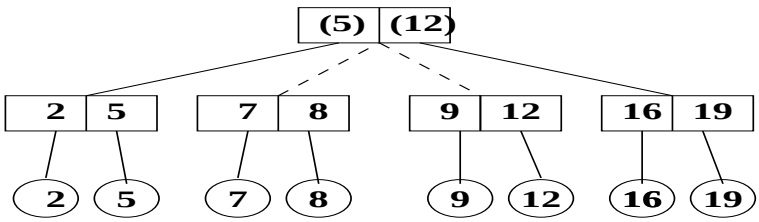
Lisäys (2, 3)-puuhun:

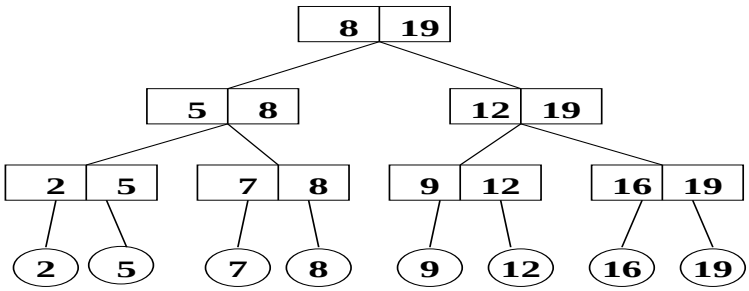
Lisättävä solmu viedään paikalleen lehdeksi.

Jos isäsolmulle tulee 4 jälkeläistä, se jaetaan kahtia; jos isoisälle tulee 4 jälkeläistä, se jaetaan kahtia; ... jne juureen saakka. (Voidaan joutua tekemään uusi juurisolmu.)

Esimerkki: lisäys 2-3 -puuhun







Poisto: käänteisessä järjestyksessä:

Jos isäsolmulle jää vain 1 jälkeläinen, se yhdistetään jonkin veljen kanssa; Jos isoisälle jää vain 1 jälkeläinen, se yhdistetään jonkin veljen kanssa; ... jne juureen saakka.

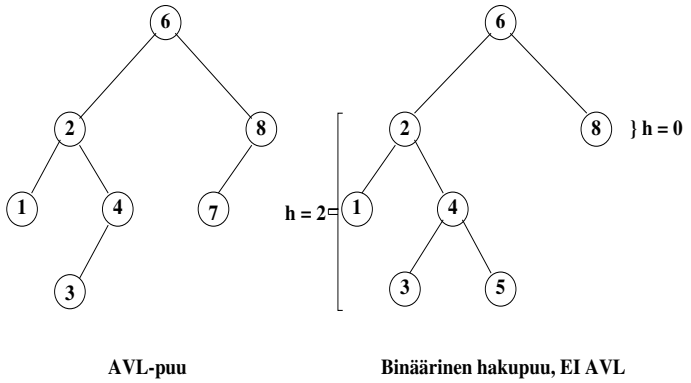
Lause. (2, 3)-puurakenteella voidaan mikä tahansa n:n member-, insert-, delete- ja min-operaation jono toteuttaa ajassa $O(n\ log\ n)$.□

2.4.2 AVL - puut

Adelson - Velskij & Landis 1962

Tasapainoehto: hakupuun jokaisessa solmussa saa vasemman ja oikean alipuun korkeusero olla enintään 1.

Esimerkki.



Tieto paikallisesta tasapainotilanteesta voidaan tallettaa kuhunkin solmuun {+1, 0, -1} -arvoisella muuttujalla.

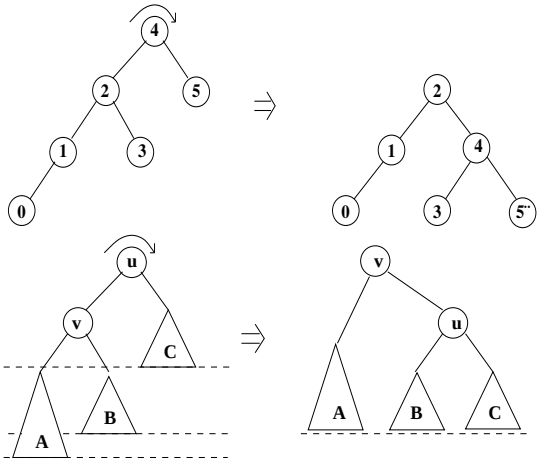
Suunnitellaan Insert- ja Delete-operaatiot niin, että tasapainoehto säilyy.

Puun kierrot

Insert-operaatio voi aiheuttaa tasapainossa olevaan AVL-puuhun 2:n suuruisen tasapainovirheen johonkin kohtaan lisätystä solmusta juureen johtavalla polulla.

Virhe voidaan korjata *kiertämällä* puu tasapainoon sen kohdalla.

Esimerkki. Oikea kierto:



Huom:

Kierto säilyttää solmujen hakupuujärjestyksen.

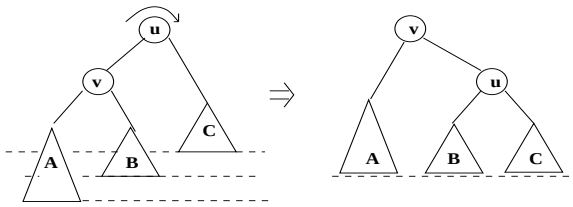
Kierto on paikallinen operaatio($\Rightarrow$ vakiokustannus).

Kierto tarvitsee tehdä enintään kerran kunkin Insert - operaation kohdalla.

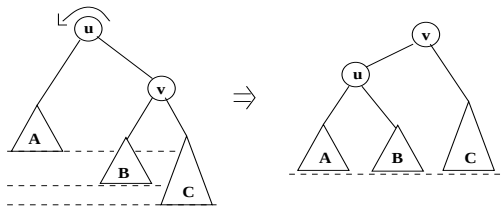
$\Rightarrow$ Kunkin Insert - operaation kustannus on $O(log\ n)$
(n - solmuisen AVL- puun syvyys on $\leq 1,44\ log_2 n$)(HT).

Kiertotyypit (jokaisessa u on solmu, jolla virheelliset alipuut)

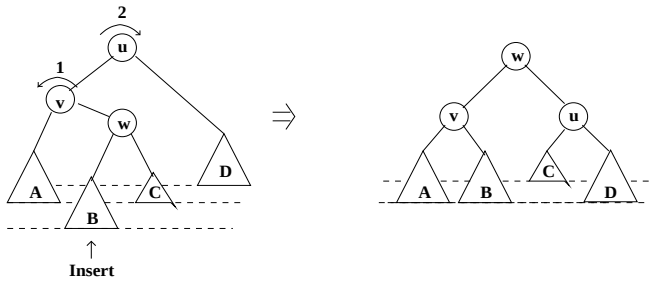
1. Oikea kierto (tehty lisäys A-puuhun; ennen yhtä tasoa matalampi)



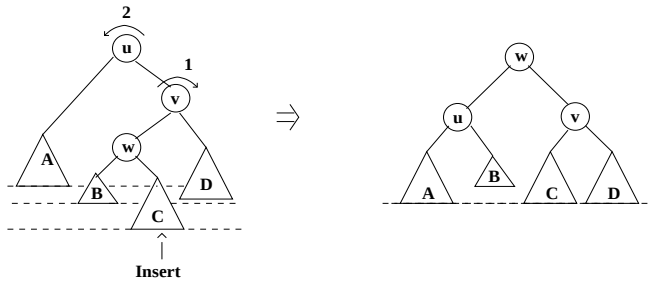
2. Vasen kierto (tehty lisäys C-puuhun; ennen yhtä tasoa matalampi)



3. Vasen-oikea kaksoiskierto (B-puu lisäyksen jälkeisessä korkeudessa; aiemmin yhtä tasoa matalampi)



4. Oikea-vasen kaksoiskierto (C-puu lisäyksen jälkeisessä korkeudessa; aiemmin yhtä tasoa matalampi)



Poisto AVL-puusta (HT).

AVL-puiden arviointia

Lause. AVL - puiden avulla voidaan mikä tahansa n Member-, Insert-, Delete- ja Min-operaation jono toteuttaa ajassa $= O(n\ log\ n)$.□

Käytännössä kuitenkin AVL-puiden heikkous on, että operaatiot vaativat puun edestakaisen läpikäynnin ylhäältä alas ja takaisin ylös. Esimerkiksi punamustilla puilla yksi läpikäynti riittää, mutta operaatiot ovat mutkikkaampia.

2.4.3 Itsesäätävät binääriset hakupuut (Splay-puut)

Idea: puun tasapainoa ylläpidetään "laiskasti": kulloinkin viitattu solmu tuodaan kierroilla puun juureen niin, että muidenkin samalla polulla olevien solmujen syvyys pienenee (tai ei ainakaan kasva).

$\Rightarrow$ Syvät (=kalliit) viittaukset tasapainottavat puuta.

$\Rightarrow$ m operaation jonon tasattu vaativuus $O(m\ log\ n)$.

Etuja:

-Yksinkertainen tietorakenne, yksinkertaiset algoritmit
-Ei tasapainoinformaatiota, eikä sen ylläpitoa
-Tomintatakuu mille tahansa operaatiojonolle

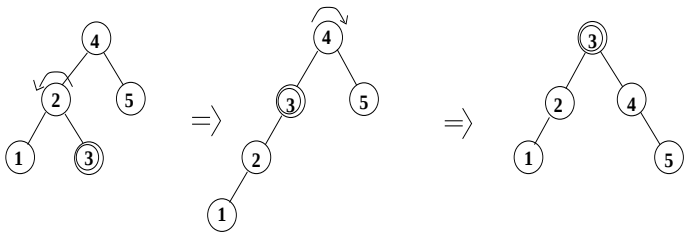
Haittoja:

-Pahin operaation yksittäiskustannus silti $O(n)$.
-Puun rakenteen jatkuva muuttaminen lisäkustannus $\rightarrow$ satunnaisessa ympäristössä muut algoritmit tehokkaampia.

ENSIMMÄINEN TASAPAINOTUSIDEA (ei toimi)

Käytetään tavallisia AVL-kiertoja:

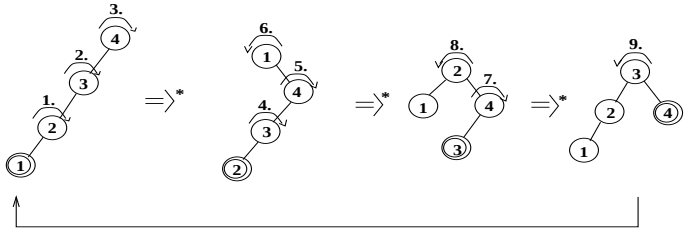
Esim. Viitattu solmuun 3.



Solmua 3 nostettaessa ovat nyt kuitenkin solmut 4 ja 5 laskeneet syvemmälle puuhun.

Tämä vaikutus voi kertautua niin, että m operaatiota n-solmuisessa puussa vaatii ajan $O(m \cdot n)$.

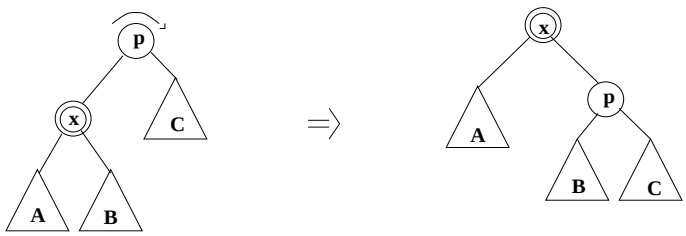
Esim.



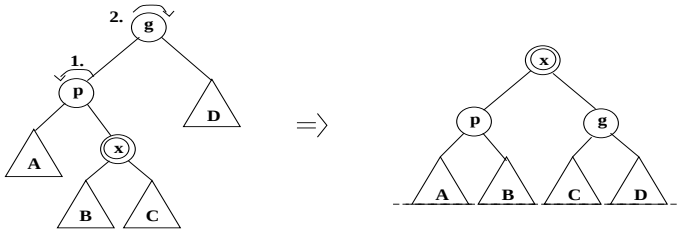
PAREMPI IDEA (Sleator& Tarjan 1985)

Kolme kiertotyyppiä (+symmetriset tapaukset):

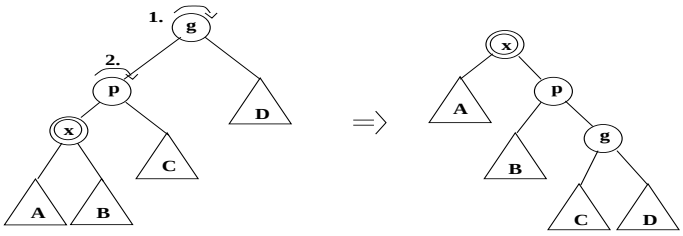
1. Sik (käytetään vain välittömästi juuren alla):



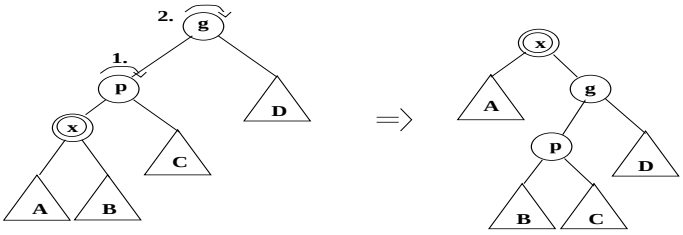
2. Sik-sak (kuten AVL - puissa)



3. Sik-sik (vain splay-puissa):



(Vrt. AVL-tyyppinen juuren kierto:)



Splay- puiden päivitysoperaatiot (Insert, Delete) HT. (Helppoja, ideana on aina operaation yhteydessä ensin nostaa solmu puun juureksi.)

2.4.4 Splay-puiden analyysi

Tasattu analyysi: halutaan osoittaa, että mielivaltainen m operaation jono n solmun puussa aiheuttaa vain $O(m\ log\ n)$ kiertoa.

$\Rightarrow$ Tarvitaan potentiaalifunktio Φ, jonka arvo kasvaa enintään $O(log\ n)$ verran kunkin operaation yhteydessä, mutta vaihtelu riittää silti tasaamaan operaatioiden kustannukset. (Huom. yksittäisen operaation kustannus voi olla $\Theta(n)$!)

Toimiva valinta puulle T:

$$\Phi(T) = \sum_{i \in T} log_2 S(i)$$

missä $S(i)$ = solmun i jälkeläisten määrä.

Sanotaan, että

$$R(i) \doteq log_2 S(i)$$

on solmun $i \in T$ *ranki*.

Esim. juuren ranki=$log_2|T|$, so. täydellisessä binääripuussa puun korkeus.

Huom.: splay-puiden kierto-operaatiot muuttavat vain niihin osallistuvien solmujen x, p, g rankeja.

Määritellään siis splay - puulle T potentiaalifunktio

$$\Phi(T) = \sum_{i \in T} R(i), \quad R(i) = log_2 S(i), \quad R(T) = R(root(T))$$

Kiertojen kustannukset määritellään:

$$c(sik) = 1,\ c(siksak) = c(siksik) = 2$$

ja tasatut kustannukset operaatiojonossa $op_1, ..., op_m$

$$\hat{c}(op_j) = c(op_j) + \Phi(T_j) - \Phi(T_{j-1}).$$

Lause. Splay-puun T minkä tahansa solmun x juureen nostamisen tasattu kustannus on enintään $3(R(T) - R(x)) + 1 = O(log\ n)$.

Todistus. Lasketaan yhteen x:n nostamiseksi tarvittavien kiertojen tasatut kustannukset.

Merkitään: $R(u), S(u), R'(u), S'(u)$ solmun u ranki ja alipuun koko ennen ja jälkeen tarkasteltavan kierron.

Osoitetaan, että solmun x sik-kierron tasattu kustannus on

$$\leq 3(R'(x) - R(x)) + 1$$

ja sekä siksak- että siksik-kierron tasattu kustannus

$$\leq 3(R'(x) - R(x)),$$

jolloin kiertojen kustannusten yhteenlasku tuottaa "teleskooppisumman"

$$\sum_{j=1}^{k} \hat{c}(op_j) \leq 1 + 3(R^k(x) - R^{k-1}(x)) + 3(R^{k-1}(x) - R^{k-2}(x)) + ...$$

$$+3(R'(x) - R^0(x)) = 1 + 3(R^k(x) - R^0(x)) = 1 + 3(R(T) - R(x)).$$

KIERTOJEN TASATUT KUSTANNUKSET

1. Sik: c(sik)=1

$$\begin{aligned} \hat{c}(sik) &= 1 + R'(x) + R'(p) - R(x) - R(p) \\ &\leq 1 + R'(x) - R(x) \qquad |S'(p) \leq S(p) \\ &\leq 1 + 3(R'(x) - R(x)) \qquad |R'(x) \geq R(x) \end{aligned}$$

2. Siksak: c(siksak)=2

$$\begin{aligned} \hat{c}(siksak) &= 2 + R'(x) + R'(p) + R'(g) - R(x) - R(p) - R(g) \\ &\leq 2 + R'(p) + R'(g) - R(x) - R(p) \qquad |S'(x) = S(g) \\ &\leq 2 + R'(p) + R'(g) - 2R(x) \qquad |S(p) \geq S(x) \end{aligned}$$

Koska $S'(p) + S'(g) \leq S'(x)$, saadaan:

$$\begin{aligned} R'(p) + R'(g) &= log_2 S'(p) + log_2 S'(g) \\ &= log_2 S'(p) \cdot S'(g) \\ &\leq log_2[(S'(p) + S'(g))/2]^2 \qquad |\text{AGM:}\sqrt{ab} \leq \frac{a+b}{2} \\ &\leq 2log_2(S'(x)/2) \\ &= 2log_2 S'(x) - 2 \\ &= 2R'(x) - 2. \end{aligned}$$

Siten

$$\begin{aligned}\hat{c}(siksak) &\leq 2 + (2R'(x) - 2) - 2R(x) \\ &= 2(R'(x) - R(x)) \\ &\leq 3(R'(x) - R(x)) \qquad |S'(x) \geq S(x)\end{aligned}$$

3. Siksik: Kuten siksak, käyttäen (epä)yhtälöitä $R'(x) = R(g), R'(x) \geq R'(p), R(x) \leq R(p), S(x) + S'(g) \leq S'(x)$

Yksityiskohdat HT.□

2.5 Keot

Tehokas ja yksinkertainen tietorakenne prioriteettijonojen (operaatiot insert, deletemin, min) toteuttamiseen: mikä tahansa n operaation jono ajassa $O(n\ log\ n)$.

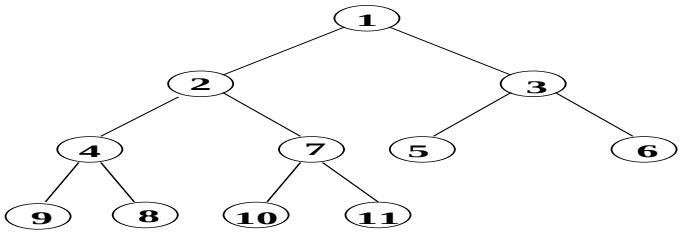
Myös perusta tehokkaalle ($O(n\ log\ n)$) "heapsort- lajittelualgoritmille.

Määritelmä: Keko on täysi, nimetty binääripuu, jonka solmunimet $v[i]$ täyttävät ehdon:

$$\text{solmu } i \text{ on solmun } j \text{ isä } \Rightarrow v[i] \leq v[j].$$

Lisäksi vaaditaan, että puun jokainen taso on täysi, alinta mahdollisesti lukuunottamatta. Alin taso on täytetty vasemmasta reunasta alkaen.

Esimerkki: Joukon {1,2,...,11} eräs kekoesitys:



n alkion keko voidaan kätevästi tallettaa taulukkoon $v[1..n]$ siten, että keon juuri sijoitetaan alkioon 1, ja alkion i kaksi jälkeläistä solmuihin $2i$ ja $2i+1$.

Esimerkki: Edellinen keko taulukkona:

v	1	2	3	4	7	5	6	9	8	10	11	
	1	2	3	4	5	6	7	8	9	10	11	

2.5.1 Keko-operaatioiden toteuttaminen

Oletetaan, että keko on talletettu taulukon $v[1..n]$ alkioihin $v[1..imax]$, $0 \leq imax \leq n$, tunnetulla tavalla.

Merkitään: *father(i)* $= \lfloor i/2 \rfloor$, *lson(i)*=$2i$, *rson(i)*=$2i$+1.

```
procedure siftup(i);                    -nosta arvo v[i] oik. paikalle keossa
          while i > 1 and v[i] < v[father(i)] do
          begin v[i] ↔ v[father(i)];
                i := father(i)
          end.

procedure siftdown(i);                  -laske arvo v[i] oik. paikalle keossa
begin     if      rson(i) ≤ imax and v[rson(i)] ≤ v[lson(i)] then
                  minson := rson(i)
          else    minson := lson(i)
          while   lson(i) ≤ imax and v[i] > v[minson] do
          begin   v[i] ↔ v[minson];
                  i := minson;
                  if rson(i) ≤ imax and v[rson(i)] ≤ v[lson(i)] then
                        minson := rson(i)
                  else  minson := lson(i)
          end end.

procedure insert(a);
          if      imax = n then fail else
          begin   imax := imax + 1;
                  v[imax] := a;
                  siftup(imax)
          end.

procedure heapify;  –järjestä taulukko v kekojärjestykseen
          for i := imax downto 1 do
                  siftdown(i)


function Deletemin;
          if imax = 0 then fail else
          begin   a := v[1];
                  v[1] := v[imax];
                  imax := imax − 1;
                  siftdown(1);
                  return a
          end.
```

2.5.2 Keko-operaatioiden analyysi

n-alkioisessa keossa kukin *siftup*- ja *siftdown*-operaatio vie ajan $O(log\ n)$.

$\rightarrow$ Siis myös *Insert*- ja *Deletemin*- operaatioiden aikaraja $O(log\ n)$.

heapify-operaation triviaali aikaraja $O(n\ log\ n)$; itseasiassa $O(n)$ riittää:

n-alkion keossa on korkeudella h lehdistä lukien $\leq n/2^h$ alkiota.

Kunkin korkeudella h olevan alkion kohdalla on *siftdown*-operaation kustannus $\leq c(h+1)$ (c vakio).

Kokonaiskustannus siis enintään:

$$\sum_{h=0}^{\lfloor log_2 n \rfloor} \frac{n}{2^h} \cdot c(h+1) \leq 2cn \cdot \sum_{h=0}^{\infty} \frac{h+1}{2^{h+1}} = 4cn = O(n).$$

2.5.3 Kekolajittelu (heapsort)

Idea: järjestetään lajiteltava taulukko ensin keoksi (*heapify*) ja poistetaan sitten alkiot järjestyksessään yksi kerrallaan (*Deletemin*).

Toteutus:

procedure $Heapsort(A[1..n])$;
    $heapify(A[1..n])$;
    for $k := n-1$ **downto** 1 **do**
        { $A[1] \leftrightarrow A[k+1]$; $siftdown$(A, 1, k)}
        missä $siftdown(A, 1, k)$ on parametrisoitu $siftdown$

Huom: tämä versio lajittelee alkiot käänteiseen suuruusjärjestykseen; järjestys on helppo vaihtaa.

Aikavaativuus:

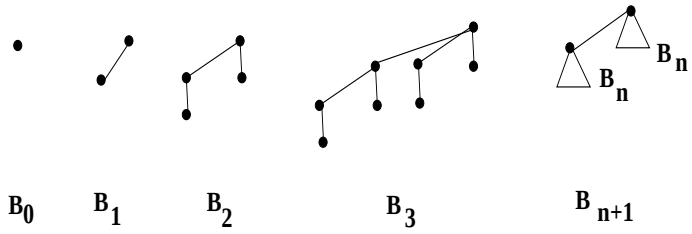
$$\underbrace{O(n)}_{\text{heapify}} + \sum_{k=n-1}^{1} \underbrace{O(log\ k)}_{\text{siftdown}} = O(n) + n \cdot O(log\ n) = O(n\ log\ n).$$

2.6 Binomimetsät

Tietorakenne yhdistyvien prioriteettijonojen (operaatiot insert, delete, min, union) toteuttamiseen.

Määritelmä: Binomipuu B_0 on puu, jossa on yksi solmu. Binomipuu B_{n+1} muodostuu kahdesta B_n-puusta asettamalla toisen juuri toisen juuren pojaksi.

Esimerkki:



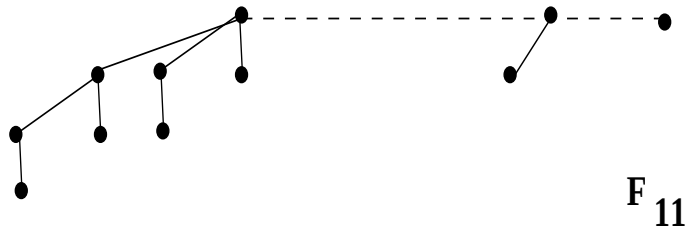
Puun B_n ominaisuuksia:

- solmuja 2^n kpl
- korkeus n
- syvyydellä k on $\binom{n}{k}$ solmua

Olkoon sitten $n = \sum_i b_i 2^i$.

Binomimetsä F_n koostuu juuristaan yhteen linkitetyistä puista $\{B_i | b_i = 1\}$.

Esimerkki: $11_{10} = 1011_2$



Metsässä F_n on enintään $\lfloor log_2 n \rfloor + 1$ puuta.

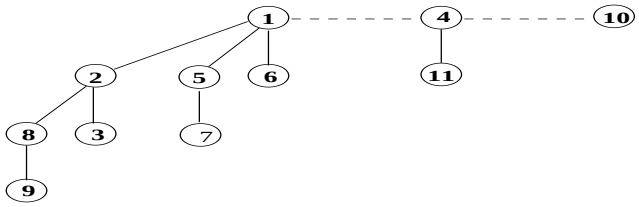
Järjestetyn joukon toteuttaminen binomimetsänä:

Olkoon S n alkion järjestetty joukko. S:n alkiot talletetaan metsän F_n solmuihin siten, että seuraava *kekoehto* toteutuu:

Merkitään: $l(u) \in S \sim$ solmu u sisältämä arvo.

Tällöin: solmu u on solmun v isä $\Rightarrow l(u) \leq l(v)$

Esimerkki: Joukon $S = \{1, 2, \ldots, 11\}$ talletus binääripuuna

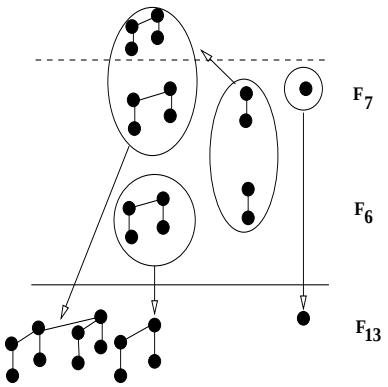


2.6.1 Binomimetsien operaatiot

1. union $(F_i; F_j)$

Jos $i = j = 2^p$ jollakin p (jolloin $F_i \cong F_j \cong B_p$), puut yhdistetään puuksi B_{p+1} asettamalla "ylemmäksi"se, jonka juuren arvo on pienempi. Muuten edetään kuten binäärilukujen yhteenlaskussa.

Esimerkki: $F_7 + F_6$



Aikavaativuus $O(log\ i + log\ j)$

2. insert (a, F_m)

Toteutetaan kuten $union(\{a\}, F_m) \Rightarrow$ aikavaativuus $O(log\ m)$.

n lisäystä m-alkioiseen metsään vaatii kuitenkin vain ajan $O(n + log\ m)$.

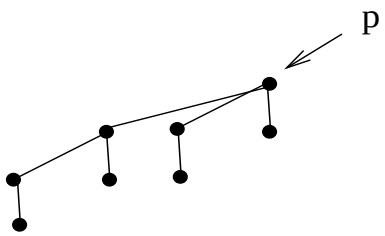
(vrt. binäärilaskurin tasattu analyysi, luku 1.5)

3. min (F_m)

Pienin alkio on jonkin F_m:n puun juuressa $\Rightarrow$ aikavaativuus $O(log\ m)$.

4. delete (p, F_m)

Olkoon poistettavana puun B_i juuri. Tällöin B_i hajoaa metsäksi F_{2^i-1}, joka voidaan yhdistää jäljelle jääviin puihin operaatiolla $union(F_{m-2^i}, F_{2^i-1}) \Rightarrow$ aikavaativuus $O(log\ m)$.



Muut solmut: nostetaan solmu ensin puunsa (kekonsa) juureen *siftup*-operatiolla (keinotekoinen arvo -∞) ja poistetaan sitten sieltä $\Rightarrow$ aikavaativuus $O(log\ m)$.

Itsesäätäviä tietorakenteita yhdistyville prioriteettijonoille:

- vasemmistolaiset keot (engl. leftist heap)
- vinot keot (engl. skew heap), parikeot (engl. pairing heap)

2.7 Erillisten joukkojen toteuttaminen

(union-find-algoritmi)

Käyttökelpoinen algoritmi ja hyvä analyysiesimerkki.

Algoritmi: Galler & Fischer 1964

Analyysi: Hopcroft & Ullman 1973, Tarjan 1975.

Mielivaltaisen perusjoukon tapauksessa voidaan hyvillä tietorakenteilla (tasapainoitetut puut, keot jne.) n joukko-operaatiota toteuttaa ajassa $O(log\ n)$/ operaatio.

Jos tarkasteltava perusjoukko on pieni, päästään vieläkin parempiin suoritusaikoihin.

Tarkasteltava tilanne:

Käsiteltävänä erilliset joukot $S_1, \ldots, S_n$; sisältävät kokonaislukuja väliltä $1, \ldots, n$. Aluksi $S_i = \{i\}$ kaikilla i.

Suoritettavat operaatiot *union*, *find* (*insert*, *delete*, *member*):

union(u,v,t): $S_t \leftarrow S_u \cup S_v$, jos $S_u \cap S_v = \emptyset$; muuten ei määritelty

find(i): indeksi t s.e. $i \in S_t$.

Sovelluksia:

Alkioiden ekvivalenssiluokkien muodostaminen (esim. äärellisten automaattien minimoinnissa).

Pienimmän virittävän puun määrittäminen (Kruskalin algoritmi). Ym. verkkoalgoritmit.

Taulukkoihin perustuva ratkaisu

Taulukko $R[1..n]$: $R[i] = u \leftrightarrow i \in S_u$

find(i): **return** $R[i]$ –aika $\Theta(1)$

union(u, v, t): **for** $i = 1$ **to** n **do** –aika $\Theta(n)$
if $R[i] = u$ **or** $R[i] = v$
then $R[i] \leftarrow t$

n operaatiota ajassa $\Theta(n^2)$.

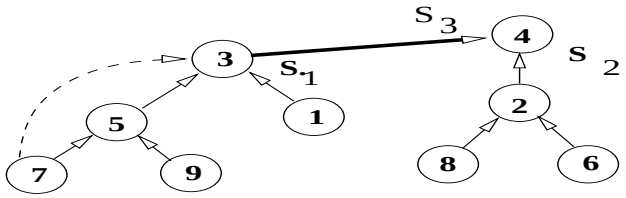
Puihin perustuva ratkaisu

Joukko S esitetään juurellisena puuna, jonka solmut vastaavat S:n alkioita.

Puut linkitetään juuria kohti. Joukon nimi (S) liitetään juureen.

Esimerkki:

$S_1 = \{1, 3, 5, 7, 9\}$ $S_2 = \{2, 4, 6, 8\}$



find(7)=1

union(1,2,3)

find(7)=3

Aikavaativuus: n operaation jonossa

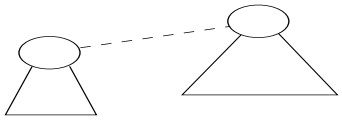
· kukin *union*-operaatio ajassa $O(1)$

· kukin *find*-operaatio keskimääräisessä ajassa $O(log\ n)$, pahimmassa tapauksessa $O(n)$

$\therefore$ n operaatiota ajassa $O(n^2)$ (pahin tapaus)

Parannus 1: puiden tasapainotus

Periaate: Pidetään kirjaa puiden solmumääristä. *Union*-operaatiossa liitetään aina pienempi puu suuremman alipuuksi. (Samankokoiset mitenpäin tahansa).



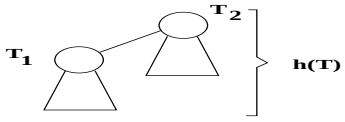
Lause. Tasapainotusmenettelyä käytettäessä voi n-solmuisen puun syvyydeksi tulla enintään $log_2 n$.

Todistus: Induktiolla puun muodostaneiden *union*-operaatioiden jonon pituuden suhteen. Merkitään

$$|T| = \text{puun T solmujen lukumäärä}$$
$$h(T) = \text{puun T syvyys}$$

Alkutilanne: T on yksisolmuinen ja siis $h(T) = 0 = log_2|T|$.

Induktioaskel: Olkoon T saatu yhdistämällä puut T_1 ja T_2



Tarkastellaan tapausta $|T_1| \leq |T_2|$. Koska induktio-oletuksen perusteella $h(T_1) \leq log_2|T_1|$ ja $h(T_2) \leq log_2|T_2|$, niin

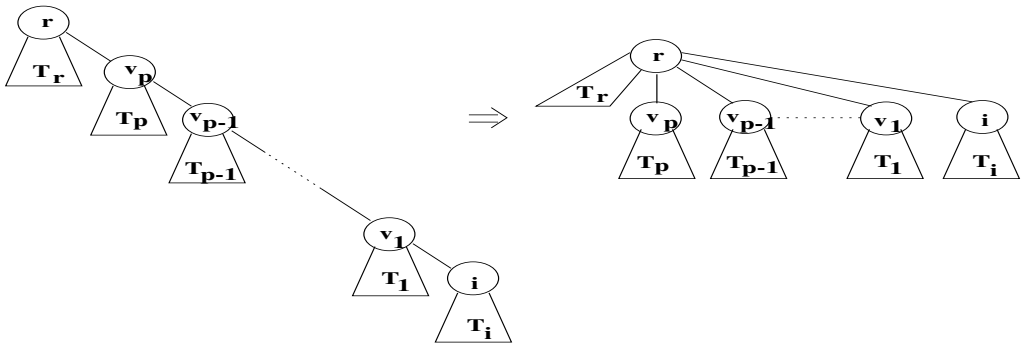
$$\begin{aligned} h(T) &= \max(h(T_2), h(T_1)+1) \\ &\leq \max(log_2|T_2|, 1+log_2|T_1|) \\ &= \max(log_2|T_2|, log_2 2|T_1|) \\ &\leq log_2(|T_1|+|T_2|) \\ &= log_2|T|.\Box \end{aligned}$$

Siten tasapainotus $\Rightarrow$ *find*-operaatiot toimivat ajassa $O(log\ n)$.

$\therefore$ Tasapainotusta käytettäessä voidaan n *union*- ja *find*-operaatiota suorittaa ajassa $O(n\ log\ n)$.

Parannus 2: polun tiivistys

Periaate: Operaation *find*(i) yhteydessä joudutaan kulkemaan polku solmusta i juureen r. Liitetään kaikki polulla tavatut solmut suoraan juuren pojiksi.



Lause 1: Jos käytetään tiivistämistä, mutta ei tasapainotusta, voidaan n *union*- ja *find*-operaatiota suorittaa ajassa $O(n\ log\ n)$. $\Box$

Määritellään:

$$\begin{cases} exp^*-1 &= 0 \\ exp^*0 &= 1 \\ exp^*k &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{k\ \text{kpl}} & ,k \geq 1 \end{cases}$$

$$log^*n = \text{pienin } k \text{ s.e. } exp^*k \geq n.$$

$$\Rightarrow log^*n \leq 6 \text{ kaikilla käytännössä esiintyvillä } n$$

Lause 2. Jos käytetään sekä tiivistämistä että tasapainotusta, voidaan n *union*- ja *find*-operaatiota suorittaa ajassa $O(n\ log^*n)$. □

2.7.1 Union-find-algoritmin analyysi

Tasattu analyysi.

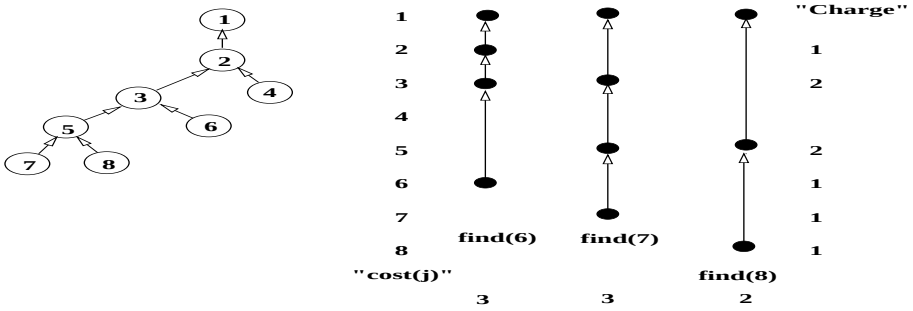
Tarkastellaan n alkion joukon *union-find*-käskyjonoa, jossa on m *find*-operaatiota (ja enintään $n-1$ *union*-operatiota). Käytetään sekä tiivistystä että tasapainotusta.

Kukin *Union*-operaatio vaatii vakioajan $\Rightarrow$ kustannus $O(n)$.

Find-operaatioiden analysoimiseksi tarkastellaan *perusmetsää* F, joka saadaan suorittamalla pelkästään operaatiojonon *union*-operaatiot (so. ei poluntiivistyksiä).

kunkin *find*-operaation kustannus on verrannollinen sen hakupolun pituuteen; analysoidaan hakupolkujen lyhenemistä perusmetsässä.

Esimerkki.



Idea: Laskutetaan solmuja niiden kautta kulkevista hakupoluista. *Find*-jonon kokonaiskustannus on O(hakupolkujen kaarien yhteismäärä).

Solmujen ryhmittely: merkitään

$$h(v) = \text{ solmun } v \text{ korkeus perusmetsässä } F;$$

$$C_i = \{v | exp^*(i-1) < h(v) \leq exp^*i\}, 0 \leq i \leq log^*n.$$

Laskutusperiaate: hakupolkujen kaaret jaetaan niiden solmujen ja *find*-operaatioiden kesken seuraavasti:

(i) kunkin hakupolun viimeinen kaari laskutetaan haun aiheuttaneelle *find*-operaatiolle;

(ii) kukin kaari $u \to v$, missä u ja v kuuluvat samaan korkeusluokkaan C_i, laskutetaan solmulle u;

(iii) kukin kaari $u \to v$, missä u ja v kuuluvat eri korkeusluokkiin, laskutetaan haun aiheuttaneelle *find*-operaatiolle.

Kaikkiaan saadaan:

(a) Kutakin *find*-operaatiota laskutetaan $1 + O(log^* n)$ yksikköä.

(b) Kutakin solmua $u \in C_i$ laskutetaan $\leq (exp^* i - exp^*(i-1))$ yksikköä, koska jokaisen kaaren $u \to v$ laskutuksen yhteydessä u:n uudeksi seuraajaksi tulee solmu v', jolla $h(v') > h(v)$. Lopulta u:n seuraaja siirtyy eri korkeusluokkaan, ja u:ta ei enää laskuteta.

Solmuja laskutetaan siten kaikkiaan yhteensä enintään

$$\sum_{i=0}^{log^* n} (exp^* i - exp^*(i-1)).|C_i| \text{ yksikköä.}$$

Arvioidaan sitten korkeusluokkien C_i kokoja:

Union-operaatioiden tasapainotuksen takia on perusmetsässä F kussakin korkeudella h olevaan solmuun juurtuvassa alipuussa vähintään 2^h solmua.

Seuraa:

$$\begin{aligned} |C_i| &\leq \sum_{h=exp^*(i-1)+1}^{exp^* i} \frac{n}{2^h} \leq \frac{n}{2^{exp^*(i-1)+1}}[1 + \frac{1}{2} + \frac{1}{4} + \ldots] \\ &= \frac{n}{2^{exp^*(i-1)}} = \frac{n}{exp^* i}. \end{aligned}$$

Siten solmujen kokonaislaskutus on enintään:

$$\begin{aligned} &\sum_{i=0}^{log^* n} (exp^* i - exp^*(i-1)) \frac{n}{exp^* i} \\ &= n \sum_{i=0}^{log^* n} \left(1 - \frac{exp^*(i-1)}{exp^* i}\right) \\ &\leq n\ log^* n. \end{aligned}$$

kaikkiaan saadaan operaatiojonon kustannukseksi:

union-operaatiot	$O(n)$
find-operaatiot	$O(m\ log^* n)$
solmut	$O(n\ log^* n)$
Yhteensä	$O((n+m)log^* n)$.

Luku 3

Verkkoalgoritmeja

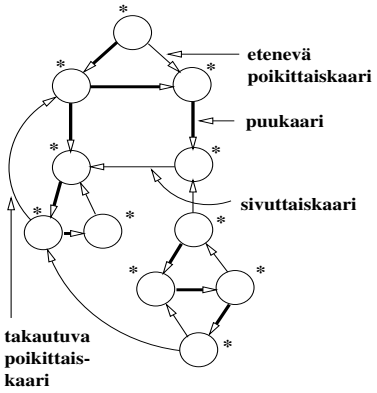
3.1 Syvyyshaku (Depth-first search)

Tärkeä verkkojen läpikäyntimenetelmä. Läpikäynti etenee pitkin verkon kaaria. Jos kaari johtaa haun ennen näkemättömään solmuun, se merkitään *puukaareksi*, muuten se on *poikittaiskaari*. Jako ei ole yksikäsitteinen, vaan riippuu kaarien valintajärjestystä. Puukaaret muodostavat verkon virittävän metsän. Suunnatussa verkossa poikittaiskaaret jakautuvat kolmeen luokkaan:

(i) etenevät kaaret: johtavat solmusta sen jälkeläisiin olematta puukaaria;

(ii) takautuvat kaaret: johtavat solmusta sen esi-isiin (mahdollisesti myös solmuun itseensä)

(iii) sivuttaiskaaret: kaaret (v, w), missä v ei ole w:n eikä w v:n jälkeläinen.



Algoritmi: suunnatun verkon syvyyshaku.

Syöte: Suunnattu verkko $G = (V, E)$, esitetty vieruslistoina $L[v], v \in V$.

Tulos: Kaikki verkon G solmut ja kaaret käydään läpi. Kaaret jaetaan puukaariin T ja poikittaiskaariin $B = E \backslash T$.

Menetelmä:

```
T :=∅
for each v ∈ V do mark[v]:=new;
for each v ∈ V do
    if mark[v]= new then search(v);

procedure search(v);
begin
    mark[v]:=old;
    for each w ∈ L[v] do
     if mark[w]= new then
     begin
       insert((v,w),T);
       search(w)
     end;
end.
```

Lause 1. Suunnatun verkon $G = (V, E)$ syvyyshaun aikavaativuus on $O(n + e)$, missä $n = |V|$, $e = |E|$.

Todistus. Algoritmin alustustoimet vaativat ajan $O(n)$. Proseduuri *search* tutkii kunkin verkon kaaren kerran ja vain kerran, koska kaaren (v, w) tarkastus muuttaa solmun w "vanhaksi"(ellei se ollut jo), ja vanhoista solmuista lähteviä kaaria ei tarkasteta edelleen.□

Verkon solmut voidaan numeroida läpikäynnin yhteydessä esi- tai jälkijärjestyksessä:

prenum[v]: solmun v numero annettuna ennen v:n käsittelyä.

postnum[v]: solmun v numero annettuna ennen v:n käsittelyn jälkeen.

Algoritmi: suunnatun verkon syvyyshaku [esi- ja jälkijärjestys].

Syöte: Suunnattu verkko $G = (V, E)$, esitetty vieruslistoina $L[v], v \in V$.

Tulos: Kaikki verkon G solmut ja kaaret käydään läpi. Kaaret jaetaan puukaariin T ja poikittaiskaariin $B = E \backslash T$.

Menetelmä:

```
T :=∅; penum := 1;  ponum := 1;
for each v ∈ V do mark[v]:=new;
for each v ∈ V do
    if mark[v]= new then search(v);

procedure search(v);
begin
    prenum[v]:= penum; penum := penum + 1;
    mark[v]:=old;
    for each w ∈ L[v] do
     if mark[w]= new then
     begin
       insert((v, w), T);
       search(w)
     end;
    postnum[v]:=ponum; ponum := ponum + 1;
end.
```

Lause 2. Jos (v, w) on syvyyshaun määrittämä suunnatun verkon sivuttaiskaari, niin $prenum[v] > prenum[w]$ ja $postnum[v] > postnum[w]$.

Todistus. Tarkastellaan vain esijärjestysnumerointia. Jos olisi $prenum[v] < prenum[w]$, niin solmu w olisi vielä "uusi"kutsun $search(v)$ hetkellä. Mutta koska oletuksen mukaan w on v:n jälkeläinen, sen pitäisi tällöin tulla tarkastetuksi kutsun $search(v)$ aikana, ja kaari (v, w) olisikin etenevä eikä sivuttaiskaari. □

Syvyyshaun sovelluksia:

Saavutettavuustestaus, syklittömyystestaus.

Topologinen lajittelu: Järjestettävä suunnatun syklittömän verkon solmut johonkin lineaarijärjestykseen, so. liitettävä niihin numerot $s(v)$ s.e. jos $(u, v) \in E$, niin $s(u) < s(v)$.

Ratkaisu: käytetään käänteistä jälkijärjestystä, so. asetetaan $s(v) = |V| - postnum[v]$. Syvyyshaku on suoritettava lähtemällä solmuista, joiden tuloaste on nolla.

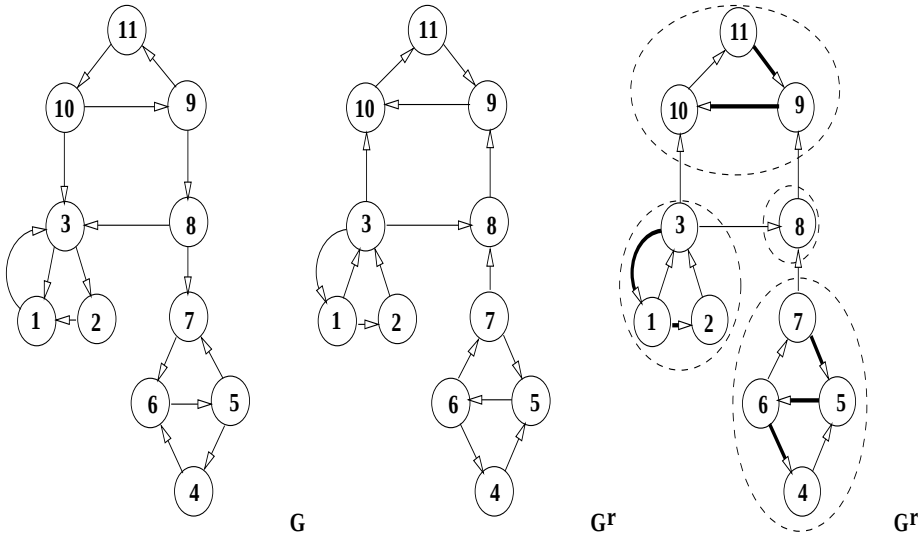
Vahvasti yhtenäiset komponentit:

Suunnatun verkon $G = (V, E)$ solmut v ja w kuuluvat samaan *vahvasti yhtenäiseen komponenttiin*, jos kummastakin johtaa suunnattu polku toiseen. [Ominaisuus on helppo todeta ekvivalenssirelaatioksi.]

Verkon jakaminen vahvasti yhtenäisiin komponentteihin on monissa sovelluksissa tärkeä tehtävä.

Algoritmi:

1. Numeroi verkon G solmut syvyyshaulla jälkijärjestyksessä.
2. Muodosta verkko G^r, jossa on samat solmut kuin verkossa G, mutta kaaret eri päin.
3. Käy verkko G^r läpi syvyyshaulla siten, että kukin hakukierros aloitetaan siitä jäljellä olevasta solmusta, jolla on suurin jälkijärjestysnumero.
4. Näin tuotetun virittävän metsän kukin puu on verkon G vahvasti yhtenäinen komponentti.



Lause. Edellinen algoritmi määrittää verkon $G = (V, E)$ vahvasti yhtenäiset komponentit ajassa $O(n + e)$, missä $n = |V|, e = |E|$.

Todistus. Aikavaativuusväite on selvä (kaksi syvyyshakua + verkon kaarien kääntäminen).

Algoritmin oikeellisuuden toteamiseksi osoitetaan, että solmut v ja w kuuluvat samaan G:n vahvasti yhtenäiseen komponenttiin, jos ja vain jos ne kuuluvat samaan G^r:n virittävän metsän puuhun.

$\boxed{\Rightarrow}$ Oletetaan, että verkossa G on polut $v \rightsquigarrow w$ ja $w \rightsquigarrow v$. Tällöin myös verkossa G^r on polut $w \rightsquigarrow v$ja $v \rightsquigarrow w$. Tämä takaa, että solmut tulevat G^r:n syvyyshaun määräämässä metsässä samaan puuhun.

$\boxed{\Leftarrow}$ Oletetaan, että v ja w kuuluvat G^r:n virittävän metsän samaan puuhun; olkoon x tämän puun juuri. Tällöin G:ssä on polut $v \rightsquigarrow x$ ja $w \rightsquigarrow x$, joten väite on todistettu, kun osoitetaan, että G:ssä on myös polut $x \rightsquigarrow v$ ja $x \rightsquigarrow w$.

Tarkastellaan esim. solmua v: oletetaan, että $v \neq x$. Koska v kuuluu x:n puuhun, on puiden muodostusperiaatteen takia oltava

$$postnum_G[x] > postnum_G[v],$$

so. G:n syvyyshaussa kutsu search(v) päättyy ennen kuin kutsu search(x). Koska toisaalta G:ssä on polku $v \rightsquigarrow x$, kutsu search(v) ei voi päättyä ennen kuin search(x) alkaa. Siten kutsujärjestyksen on oltava:

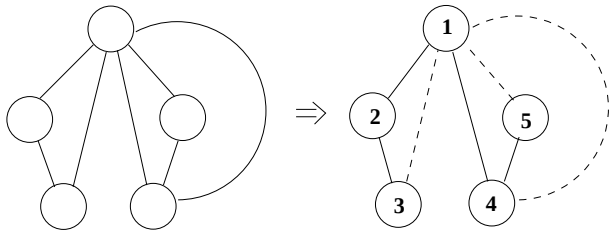
search(x)	alkaa
...	
search(v)	alkaa
...	
search(v)	päättyy
...	
search(x)	päättyy

Mutta tämä merkitsee, että G:ssä on polku $x \rightsquigarrow v$. □

3.1.1 Syvyyshaku suuntaamattomassa verkossa

Syvyyshaun tulos suuntaamattomassa verkossa on rakenteeltaan yksinkertainen: Kaaret jakautuvat puukaariin ja takautuviin kaariin, ts. kaikki poikittaiskaaret ovat takautuvia. Puukaarien muodostamaan virittävään metsään tulee kutakin verkon yhtenäistä komponenttia kohden yksi puu. (Erityisesti: jos verkko on yhtenäinen, tuloksena on yksi virittävä puu.)

Esim.



Sovellus: Verkon artikulaatiopisteet.

Olkoon $G = (V, E)$ yhtenäinen (suuntaamaton) verkko. Solmu $v \in V$ on G:n *artikulaatiopiste*, jos sen poistaminen tekee verkosta epäyhtenäisen. Verkko on *2-yhtenäinen*, jos siinä ei ole artikulaatiopisteitä. (Ilmeinen sovellus: sähkö- tai televerkon luotettavuusanalyysi.)

Artikulaatiopisteet voidaan kuvata myös verkolle syvyyshaussa saatavan rakenteen avulla:

Lemma. Olkoon $G = (V, E)$ suuntaamaton yhtenäinen verkko ja $S = (V, T)$ sen jokin syvyyssuuntainen (=syvyyshaun perusteella määrätty) virittävä puu. Solmu $a \in V$ on tällöin G:n artikulaatiopiste, jos ja vain jos

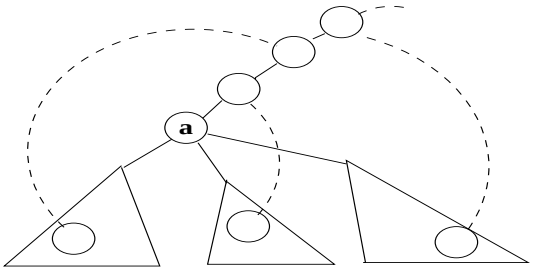
(i) a on S:n juuri ja sillä on vähintään 2 poikaa; tai

(ii) a ei ole S:n juuri, ja a:n jostakin alipuusta (T:ssä) ei johda yhtään takautuvaa kaarta a:n mihinkään aitoon esi-isään.

Todistus. Juurisolmun tapaus on helppo, joten oletetaan, että a ei ole juurisolmu.

$\boxed{\Leftarrow}$ Jos ehto (ii) on voimassa, a:n poistaminen selvästi tekee verkosta epäyhtenäisen. (Huom: G:ssä ei ole sivuttaiskaaria, joten a:n alipuut ovat erillisiä.)

$\boxed{\Rightarrow}$ Oletetaan, että ehto (ii) ei ole voimassa. Tällöin voidaan todeta, että a voidaan aina ”ohittaa” takautuvia kaaria pitkin, eikä se siis ole artikulaatiopiste. Kuva:



□

Lemman nojalla artikulaatiopisteet voidaan määrittää tutkimalla syvyyshaun yhteydessä, miten "ylös" hakupuussa T kunkin solmun alapuolelta on mahdollista päästä takautuvilla kaarilla. Tämän vuoksi solmuihin liitetään low-arvo, missä ylläpidetään tietoa siitä, kuinka korkealle meneviä (takautuvia) kaaria solmuista alkavasta alipuusta on toistaiseksi havaittu.

Määr. kaikilla $v \in V$:

low[v]=min({prenum[v]}∪
{low[w]| w on v:n poika T:ssä}∪
{prenum[w]|(v, w) on takautuva kaari.})

Algoritmi:

```
for each v ∈ V do mark[v]:=new;
pnum:=1;
for each v ∈ V do
    if mark[v]=new then searchb(v);

procedure searchb(v);
begin
    mark[v]:=old;
    prenum[v]:=pnum;
    low[v]:=pnum;
    pnum:=pnum+1;
    for each w ∈ l[v] do
        if mark[w] = new then
        begin
              searchb(w);
              if low[w] ≥ prenum[v] then
              <v on artikulaatiopiste>
              else
              low[v]:=min(low[v],low[w])
        end else
              low[v]:=min(low[v], prenum[w])
end.
```

Algoritmista voidaan laatia myös versio, joka tulostaa verkon 2-yhtenäiset komponentit. (Solmut u ja v kuuluvat samaan 2-yhtenäiseen komponenttiin, jos niiden kautta kulkee verkossa sykli.) x

3.2 Eulerin kehät ja polut

Suunnatun tai suuntaamattoman verkon G *Eulerin kehä* on G:n sykli, joka kulkee sen jokaisen kaaren kautta täsmälleen kerran. Verkon Eulerin *polku* määritellään samoin, mutta syklisyyttä ei vaadita (so. alku- ja loppusolmu voivat olla eri).

Lause. E (L. Euler 1735). suuntaamaton verkko G sisältää Eulerin kehän, jos ja vain jos

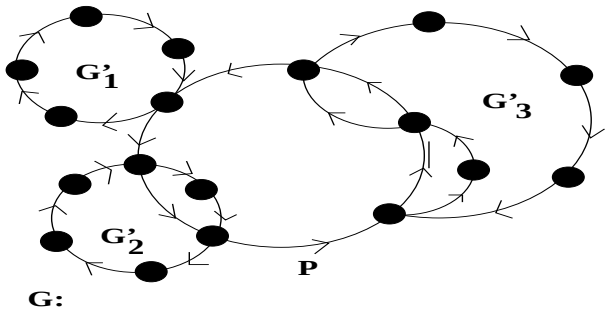
(i) G on yhtenäinen; ja

(ii) G:n jokaisen solmun asteluku on parillinen.

Tod. $\boxed{\Rightarrow}$ Selvä.

$\boxed{\Leftarrow}$ Väite tod. induktiolla verkon $G = (V, E)$ kaarten määrän $e = |E|$ suhteen. Oletetaan että väite pätee kaikilla verkoilla, joissa kaarten määrä on $< e$, ja olkoon G yhtenäinen verkko, jossa jokaisen solmun asteluku on parillinen.

Omin. (ii) $\Rightarrow$ G:ssä on ainakin yksi sykli $P \subseteq E$. Poistetaan G:stä tämä sykli ja tarkastellaan verkkoa $G' = (V, E - P)$.Verkolla G' on edelleen ominaisuus (ii). G' ei tosin ole välttämättä yhtenäinen, mutta jokainen sen komponentti $G'_1, \ldots, G'_k$ on. Induktio $\Rightarrow$ jokainen komponentti sisältää Eulerin kehän. Nämä voidaan liittäää yhteen syklin P kanssa Eulerin kehäksi verkolle G. □



Samaan tapaan voidaan todistaa seuraavat tulokset:

Lause E'. Suuntaamaton yhtenäinen verkko G sisältää Eulerin polun , jos ja vain jos G:n solmuista on paritonasteisia 0 tai 2. □

Lause E". Suunnattu yhtenäinen verkko G sisältää Eulerin polun, jos ja vain jos joko G:n kaikissa solmuissa on tuloaste=lähtöaste tai yhdessä solmussa on t.a. =l.a.-1 ja yhdessä solmussa on t.a.=l.a.+1.□

3.2.1 Suunnatun Eulerin kehän muodostaminen

Syöte: Vieruslistoina esitetty yhtenäinen suunnattu verkko $G = (V, E)$.

Tulos: Verkon G Eulerin kehä, mikäli sellainen on, kaarilistoina P.

Menetelmä: Kunkin solmun v vieruslistan $L[v]$ yhteydessä pidetään yllä osoitinta, joka osoittaa listan ensimmäisen "käyttämättömän" kaaren. Muodostettavan Eulerin kehän kaaret kerätään listaan P. Aina kun verkon läpikäynnissä tullaan solmuun , josta ei lähde yhtään käyttämätöntä kaarta, kuljetaan listaa P (so. jo muodostettua kehänosaa) pitkin eteenpäin, kunnes tullaan joko solmuun josta kehää voidaan jatkaa tai aloitussolmuun.

Aikavaativuus: Kukin G:n kaari käsitellään kahteen kertaan: ensin se merkitään vieruslistassa "käytetyksi" ja sitten P:ssä "kuljetuksi". Algoritmin aikavaativuus on siten $O(|E|)$.

```
function Euler(G);
begin
    olk. G = (V, E) esitetty vieruslistoina L[v], v ∈ V;
    emptylist(P);
    olk. v_0 ∈ V jokin aloitussolmu;
    v := v_0;
    repeat
        if listassa L[v] on "käyttämättömiä" kaaria then
        begin
            olk. e = (v, w) jokin tällainen;
            insert(e, P); {lisätään P:n käsillä olevaan kohtaan: ei vält. loppuun.}
            merkitse e "käytetyksi";
            v := w
        end else
        begin
            olk. e = (v, w) listan P ensimmäinen "kulkematon" kaari;
            {Tämä kaari alkaa välttämättä v:stä}
            merkitse e "kuljetuksi";
            v := w
        end
    until lista P on kokonaan "kuljettu";
    return P
end.
```

3.3 Pariutusongelma (Graph Matching)

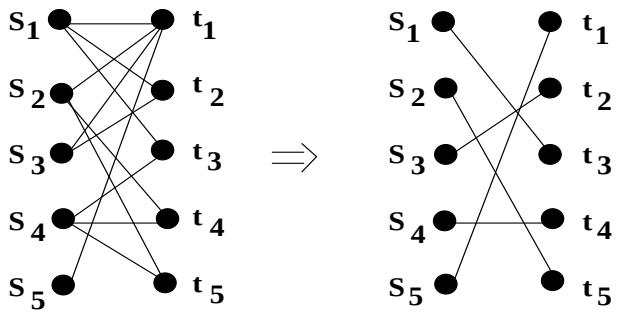
Sovellusongelma: n "palvelijan" $s_1, \ldots, s_n$ on suoritettava n "tehtävää" $t_1, \ldots, t_n$. Eri palvelijoiden soveltuvuutta eri tehtäviin kuvaa yhteensopivuusmatriisi A:

	t_1	t_2	t_3	t_4	t_5
s_1	1	1	1	0	0
s_2	1	0	0	1	1
s_3	1	1	0	0	0
s_4	0	0	1	1	1
s_5	1	0	0	0	0

$a_{ij} = 1 \sim$ palvelija s_i soveltuu tehtävään t_j.

Ratkaistava, voidaanko palvelijat ja tehtävät sovittaa yhteen, ja jos voidaan, määrättävä jokin palvelijoiden sijoittelu (engl. assignment).

Tehtävä voidaan kuvata myös verkko-ongelmana: muodostetaan palvelijoiden ja tehtävien yhteensopivuutta kuvaava kaksijakoinen verkko ja yritetään löytää sen jokin *täydellinen pariutus* (engl. perfect matching):



Määritelmä: Olkoon $G = (V, E)$ suuntaamaton verkko. Verkon G *pariutus* on kaarijoukko $M \subseteq E$ s.e. kuhunkin G:n solmuun tulee enintään yksi M:n kaari. Pariutus M on *täydellinen*, jos kuhunkin solmuun tulee tasan yksi M:n kaari (so. $|V|$ on parillinen ja $|M|=|V|/2$).

Pariutusongelma: löydettävä pariutus M s.e. $|M|$ mahdollisimman suuri.

Pariutusongelmalla on eniten sovelluksia *kaksijakoisissa* t. *puoliutuvissa* verkoissa, s.o. kun G:n solmut voidaan osittaa $V = V_1 \cup V_2, V_1 \cap V_2 = \emptyset$ s.e. $E \subseteq (V_1 \times V_2)$.

Lause (P.Hall 1935). Olkoon $G = (V_1 \cup V_2, E)$ kaksijakoinen verkko, jossa $|V_1| = |V_2|$. Merkitään solmujoukon $U \subseteq V_1$ naapureiden joukkoa $N(U)$:lla:

$$N(U) = \{v \in V_2 | u \in U, (u,v) \in E\}.$$

Tällöin verkossa G on täydellinen pariutus, jos ja vain jos jokaisella $U \subseteq V_1$ on $|N(U)| \geq |U|$.

Todistus.

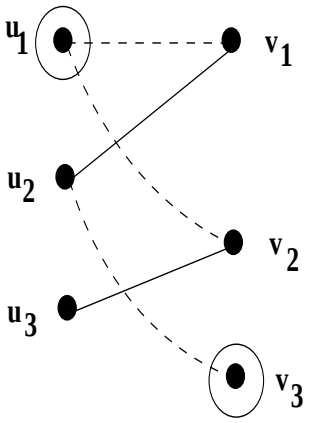
$\boxed{\Rightarrow}$ Selvä.

$\boxed{\Leftarrow}$ Oletetaan, että kaikilla $U \subseteq V_1$ on $|N(U)| \geq |U|$.

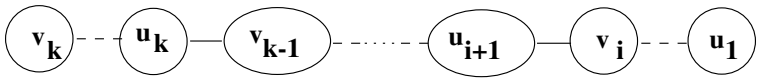
Olkoon M verkon G pariutus, jolla $|M| < |V_1|$. Tällöin on välttämättä solmuja $u \in V_1$, $v \in V_2$, joille ei ole vielä määrätty paria M:ssä. Yritetään kasvattaa M:ää saamalla kaksi tällaista vapaata solmua sen piiriin. (Tätä varten voidaan joutua uudelleen järjestelemään M:n vanhoja pareja.)

Olkoon $u_1 \in V_1$ jokin vapaa solmu. Oletuksen mukaan on $|N(\{u_1\})| \geq |\{u_1\}| = 1$, joten u_1:llä on jokin naapuri $v_1 \in V_2$. Jos myös v_1 on vapaa, pariutusta voidaan täydentää suoraan kaarella (u_1, v_1).

Muussa tapauksessa olkoon $u_2 \in V_1$ solmun v_1 pari M:ssä. Oletuksen mukaan on jälleen $|N(\{u_1, u_2\})| \geq 2$, joten olkoon $v_2 \neq v_1$ solmujoukon $\{u_1, u_2\}$ jokin uusi naapuri. Jos v_2 on vapaa, konstruktio päättyy, muuten... jne.



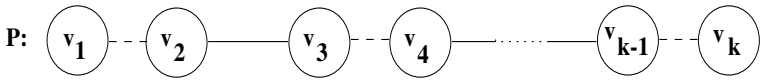
Konstruktio päättyy, kun löytyy vapaa solmu $v_k \in V_2$. Tästä solmusta "takaperin" u_1:een edeten voidaan muodostaa vuoronperään M:ään kuulumattomista ja kuuluvista kaarista polku P:



Vaihtamalla polulla P kaaret "toisinpäin" saadaan M:ää kasvatetuksi. □

Olkoon M verkon G pariutus. Verkon G polku $P = (v_1, \ldots, v_k), k > 1$, on

M:n *täydennyspolku*, jos sen joka toinen kaari on M:ssä ja v_1 ja v_k ovat *vapaita*, so. niihin ei tule yhtään M:n kaarta:



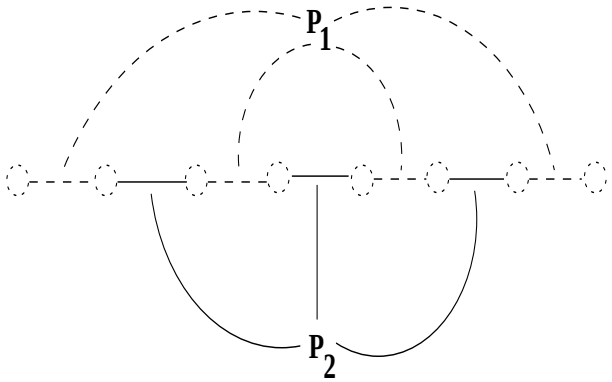
Määritelmä: Kaarijoukkojen M ja N *symmetrinen* erotus on

$$M \oplus N = (M \backslash N) \cup (N \backslash M).$$

Lemma 1. Olkoon M pariutus ja P sen täydennyspolku. Tällöin myös $M \oplus P$ on pariutus ja $|M \oplus P| = |M| + 1$.

Todistus. Olkoon

$$P = P_1 \cup P_2, P_1 \subseteq \bar{M}, P_2 \subseteq M.$$



Tällöin

$$M \oplus P = (\underbrace{M \backslash P_2}_{\text{pariutus}}) \cup P_1 \quad \underbrace{\qquad\qquad}_{\text{pariutus}}$$

Koska $|P_1| = |P_2| + 1$, on $|M \oplus P| = |M| + 1$.□

Yleiselle pariutusongelmalle saadaan näin periaatealgoritmi:

1. Aseta $M := \emptyset$.
2. Etsi M:lle täydennyspolku P.[Mutta miten?]
3. Aseta $M := M \oplus P$.

4. Toista askelia 2 ja 3 kunnes täydennyspolkua ei enää löydy. Tällöin M on maksimaalinen pariutus.

Algoritmin oikeellisuuden takaa seuraava tulos:

Lemma 2. Olkoon M ei-maksimaalinen pariutus verkossa $G = (V, E)$. Tällöin M:llä on täydennyspolku.

Todistus: Olkoon N verkon G pariutus, jolla $|N| > |M|$. Osoitetaan, että tällöin G:n aliverkko $G' = (V, M \oplus N)$ sisältää M:n täydennyspolun.

Koska M ja N ovat pariutuksia, tulee kuhunkin solmuun $v \in V$ enintään 2 joukon $M \oplus N$ kaarta. Siten kukin verkon G' yhtenäinen komponentti on joko

(a) erillinen solmu; tai

(b) sykli, jonka kaaret ovat vuorotellen joukoista $M \backslash N$ ja $N \backslash M$; tai

(c) polku, jonka kaaret ovat vuorotellen joukoista $M \backslash N$ ja $N \backslash M$;

Merkitään $S = M \cap N$; tällöin on

$$M \backslash N = M \backslash S, N \backslash M = N \backslash S, M \oplus N = (M \backslash S) \cup (N \backslash S).$$

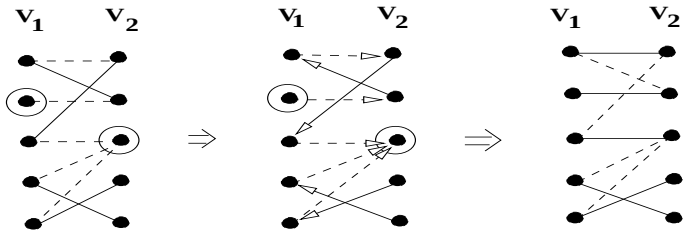
Koska $|N| > |M|$, on myös $|N \backslash S| > |M \backslash S|$. Siten joukossa $M \oplus N$ on enemmän kaaria joukosta $N \backslash M$ kuin joukosta $M \backslash N$, ja verkossa G' on oltava ainakin yksi (c)-tyyppinen komponentti. Tämä on etsitty M:n täydennyspolku. □

Täydennyspolkujen löytäminen yleisissä verkoissa on hankala tehtävä. Kaksijakoisissa verkoissa voidaan kuitenkin soveltaa esim. syvyyshakua seuraavasti:

Olkoon M jokin verkon $G = (V_1 \cup V_2, E)$ pariutus. Suunnataan verkon kaaret niin, että M:ään kuuluvat suuntautuvat V_2:sta V_1:een ja muut V_1:stä V_2:een. Tällöin M:n täydennyspolkuja ovat täsmälleen ne suunnatut polut, jotka johtavat jostakin V_1:n vapaasta solmusta johonkin V_2:n vapaaseen solmuun. Tällainen polku voidaan löytää syvyyshaulla ajassa $O(|V| + |E|)$.

Koska maksimaalisessa pariutuksessa on enintään |V|/2 kaarta, saadaan näin toteutetun algoritmin kokonaisvaativuudeksi

$$O(\frac{|V|}{2} \cdot (|V| + |E|)) = O(|V| \cdot |E|), \text{kun } |E| \geq |V|.$$



Käyttämällä täydennyspolkujen etsinnässä syvyyshaun sijaan leveyshakua ja täydentämälläuseita polkuja yhdellä haulla saadaan algoritmi, jonka vaativuus on $O(\sqrt{|V|} \cdot |E|)$. [Hopcroft & Karp 1973]

Myös yleisille verkoille tunnetaan ajassa $O(\sqrt{|V|} \cdot |E|)$ toimiva pariutusalgoritmi [Micali & Vazirani 1980].

Luku 4

Approksimointialgoritmit

4.1 Approksimointialgoritmit

Jos $P \neq NP$, niin NP-täydellisten ongelmien ratkaisemisessa täytyy tyytyä johonkin epätäydelliseen menetelmään.

Vaihtoehtoja:

- eksponentiaaliset algoritmit: Voivat toimia hyvin pienillä tapauksilla tai keskimäärin tai "käytännössä" (esim. simplex - tosin lin. ohj. $\in P$)
- Polynomiset erikoistapaukset: Esim. $2SAT \in P$
- Satunnaisalgoritmit: Hyvä idea, mutta ei luultavasti auta NP-täydellisten ongelmien ratkaisussa. Voidaan osoittaa, että jos jokin NP-täydellinen ongelma ratkeaa satunnaisalgoritmilla polynomisessa ajassa, niin "melkein" $P = NP$.
- Approksimointialgoritmit: NP-täydellisen optimointiongelman tapaukselle määritetään polynomisessa ajassa ratkaisu, joka on enintään k kertaa optimaalista huonompi, jollakin vakiolla k. Joskus voi olla myös $k = k(|x|)$, tapauksen koon suhteen hitaasti kasvava funktio.

Määritelmä:

Optimointiongelma on kolmikko

$$\Pi = (D, S, c),$$

missä

$D \sim$ ongelman tapaukset

$S \sim$ kelvolliset ratkaisut (oikeastaan vastaukset):

kuhunkin $x \in D$ liittyy joukko $S(x) \subseteq S$

$c \sim$ kustannusfunktio, $c : D \times S \to \mathbb{R}^+$.

Oletetaan seuraavassa, että Π on minimointiongelma.

Olkoon $x \in D$. Ratkaisu $s^* \in S(x)$ on optimaaalinen (minimaalinen), jos:

$$c(x, s^*) \leq c(x, s) \qquad \forall s \in S(x). \qquad [max : c(x, s^*) \geq c(x, s) \qquad \forall s \in S(x)].$$

Merkitään $c^*(x) = c(x, s^*)$, missä s^* on optimaalinen.

Ratkaisun $s \in S(x)$ hyvyys t. suhteellinen virhe on

$$r(x, s) = \frac{c(x, s) - c^*(x)}{c^*(x)}. \qquad [max : r = \frac{c^*(x) - c(x, s)}{c(x, s)}]$$

Huom: aina on $r(x, s) \geq 0$, ja $r(x, s) = 0 \Leftrightarrow s$ on optimaalinen.

Ongelman Π approksimointialgoritmi on mikä tahansa polynomisessa ajassa toimiva algoritmi A, jolla on ominaisuus

$$A(x) \in S(x) \qquad \forall x \in D.$$

Approksimointialgoritmi A on ϵ-approksimointialgoritmi, $\epsilon \geq 0$, jos

$$r(x, A(x)) \leq \epsilon \qquad \forall x \in D.$$

Esimerkki 1.

Solmupeite $VC = (D, S, c)$, missä

$D =$ kaikkien suuntaamattomien verkkojen joukko;

$S(G) =$ verkon G kaikkien solmupeitteiden C joukko;

$c(G, C) = |C|$

Eräs approksimointialgoritmi:

function VC-approx(G);
begin
olk. $G = (V, E)$;
$C := \emptyset$;
while $E \neq \emptyset$ **do**

begin
(1) (u, v):= jokin E:n kaari;
C:=$C \cup$ {u,v};
E:=E \{ kaikki u:hun tai v:hen päättyvät kaaret}
end;
return C
end.

Algoritmi selvästi toimii polynomisessa ajassa ja tuottaa jonkin solmupeitteen $C \in S(G)$. Jos merkitään

$$E_a = \{\text{algoritmin askeleessa (1) valitut kaaret}\},$$

on selvästi $|C| = 2|E_a|$.

Toisaalta minkä tahansa G:n solmupeitteen – erityisesti optimipeitteen C^* – täytyy sisältää kustakin joukon E_a kaaresta ainakin toinen pää. (Huom. E_a:n kaaret ovat erillisiä.) Siten

$$|C^*| \geq |E_a| = \frac{1}{2}|C|$$

ja

$$r(G, C) = \frac{|C| - |C^*|}{|C^*|} \leq 1,$$

ts. kyseessä on 1-approksimointialgoritmi. (tekee enintään 100% virheen).

Parhaalla tunnetulla VC-ongelman approksimointialgoritmilla saavutetaan

$$|C| \leq \left(2 - \frac{log\ log\ |V|}{log\ |V|}\right)|C^*|.$$

Esimerkki 2. TSP

Merkitään $\Delta TSP = TSP$-ongelma rajoitettuna verkkoihin $G = (V, E, c)$, joissa on voimassa kolmioepäyhtälö

$$c(u, v) \leq c(u, w) + c(w, v) \qquad \forall u, v, w \in V.$$

Myös ΔTSP-ongelma on NP-täydellinen. (Standardipalautuksella $HC \leq_m^p$ TSP voidaan tuottaa verkkoja, joissa Δ-epäyhtälö on voimassa, HC)

Eräs approksimointialgoritmi:

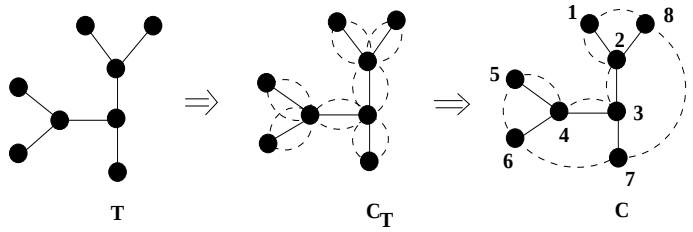
Syötteellä $G = (V, E, c)$:

(1) muodosta G:n pienin virittävä puu T;

(2) muodosta T:stä sykli C_T "kiertämällä T kahdesti";

(3) "oikaise" C_T Hamiltonin kehäksi C.

Esimerkki



Huom: reitti C vastaa puun T solmujen esijärjestettyä numerointia.

Selvästi on $c(C_T) = 2c(T)$, ja $c(C) \leq c(C_T)$, koska Δ-epäyhtälön takia oikaiseminen ei voi pidentää reittiä.

Toisaalta mistä tahansa G:n reitistä – erityisesti optitmireitistä C^* – saadaan yksi kaari poistamalla G:n virittävä puu. Koska T on G:n pienin virittävä puu, on siten $c(T) \leq c(C^*)$, mistä saadaan:

$$c(C) \leq 2c(T) \leq 2c(C^*),$$

so. algoritmi tekee enintään 100 % virheen (eli on 1-approksimointialgoritmi).

Tästä saadaan edelleen kehittämällä parempi (itse asiassa teoreettisesti paras tunnettu) algoritmi, ns. *Christofidesin algoritmi:*

Syötteellä $G = (V, E, c)$;

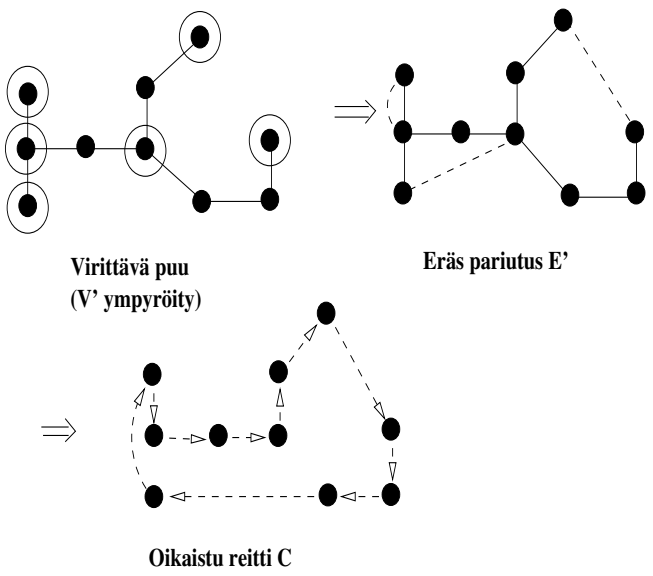
(1) muodosta G:n pienin virittävä puu T;
olkoon V' T:n paritonasteisten solmujen joukko;

(2) muodostetaan joukon V' solmujen minimipainoinen paritus E'[voidaan tehdä polynomisessa ajassa];

(3) muodostetaan Eulerin kehä C_E verkossa $(V, T \cup E')$

(4) muodostetaan reitti C "oikomalla" C_E

Esimerkki:



Selvästi on $c(C) \leq c(C_E) = c(T) + c(E')$.

Toisaalta, jos optimireitistä C^* muodostetaan oikaisemmalla pelkästään joukon V' solmut kiertävä sykli, saadaan kaksi V':n pariutusta, joista ainakin toisen kustannus on oltava $\leq \frac{1}{2}c(C^*)$.

Koska E' on minimaalinen pariutus, on kuitenkin $c(E') \leq \frac{1}{2}c(C^*)$, mistä saadaan:

$$c(C) \leq c(T) + c(E') \leq c(C^*) + \frac{1}{2}c(C^*) = \frac{3}{2}c(C^*)$$

so. algoritmi tekee enintään 50 % virheen (eli on $\frac{1}{2}$-approksimointialgoritmi).

Osoittautuu, että yleinen TSP on paljon vaikeampi kuin ΔTSP:

Lause: Jos $P \neq NP$, niin yleisellä TSP-ongelmalla ei ole polynomista k-approksimointialgoritmia millään $k \geq 0$.

Todistus: Oletetaan, että jollakin polynomisella TSP-approksimointialgoritmilla A olisi aina:

$$c(G, A(G)) \leq k \cdot c^*(G),$$

jollakin vakiolla $k \geq 0$. Tällöin Hamiltonin kehä-ongelma voitaisiin ratkaista polynomisessa ajassa ($\Rightarrow P = NP$) seuraavasti:

Muodostetaan annetusta verkosta $G = (V, E)$ TSP-ongelman tapaus $< G, d >$ asettamalla

$$d(u,v) = \begin{cases} 1, & \text{jos } (u,v) \in E, \\ (k+1) \cdot |V|, & \text{muuten} \end{cases}$$

Tässä tapauksessa on optimireitin kustannus $=|V|$, jos G:ssä on Hamiltonin kehä, ja $> k \cdot |V|$ muuten.

Algoritmin A suoritustakuun nojalla on siis verkossa G Hamiltonin kehä, jos ja vain jos

$$c(A(G, d)) \leq k \cdot |V|.$$

Tämä ehto voidaan selvästi testata polynomisessa ajassa.□

Huom: Lauseesta seuraa, että myös suunnattujen vekkojen TSP-approksimointi on vaikeata. Suunnattujen verkkojen ΔTSP-ongelman approksivoituvuudesta ei tiedetä mitään - ei tunneta algoritmeja eikä epätriviaaleja alarajoja.

Riippumaton joukko-ongelmalla voidaan todistaa samantapainen tulos kuin edellä TSP:llä, mutta todistus on erittäin paljon vaikeampi:

Lause. Jos $P \neq NP$, niin IS-ongelmalla ei ole polynomista k-approksimointialgoritmia millään $k \geq c$.

Todistus. Syvällinen [Arora et al., IEEE FOCS 1992]. □

Mielenkiintoista on, että angelmat VC ja IS ovat päätösongelmina oleellisesti samat, mutta poikkeavat näin paljon approksimoituvuudeltaan.

4.2 Approksimointiskeemat

Optimointiongelman $\Pi = (D, S, e)$ (*polynominen*) *approksimointiskeema* on algoritmi $A(x, \epsilon)$, jolla kaikilla $\epsilon > 0$ja $x \in D$ on voimassa:

(i) $A(x, \epsilon) \in S(x)$ ja $r(x, A(x, \epsilon)) \leq \epsilon$;

(ii) $time_A(x, \epsilon) \leq poly(|x|)$.

jos ehto (ii) on voimassa vahvemmassa muodossa $(ii')time_A(x, \epsilon) \leq poly(|x|, \frac{1}{\epsilon})$.

appriksimointiskeema A on *täysin polynominen.*

Toisin sanoen: approksimointiskeema takaa ongelmalle polynomisen ϵ-approksimaation millä tahansa $\epsilon > 0$, ja täysin polynominen approksimointiskeema lisäksi, että approksimoinnin hinta kasvaa vain polynomista vauhtia ϵ:n pienentyessä.

Approksimointiskeemat esitetään usein kiinteällä ϵ:n arvolla, so. muodossa $A_\epsilon(x)$, ja muokkaus muotoon $A(x, \epsilon)$ jätetään lukijalle.

Esimerkki: Repunpakkaus (KNAPSACK)

Tapaus: luonnolliset luvut $s_1, ..., s_n$, $v_1, ..., v_n$, S. | Repun tavaroiden "koot" ja "arvot" sekä "maksutus".

Kelvolliset ratkaisut: indeksijoukot $I \subset \{1, ..., n\}$ s.e.

$$\sum_{i \in I} s_i \leq S.$$

Ratkaisun kustannus (maksimoitava):

$$v(I) = \sum_{i \in I} v_i.$$

Ongelma on NP-täydellinen (helppo pal. PARTITION-ongelmasta)

Ongelma voidaan ratkaista (täsmällisesti) seuraavassa esitettävällä algoritmilla, joka taulukoi parhaat tavat muodostaa kunkin arvoinen reppu.

Algoritmi:

```
function knapsack (s[1..n], v[1..n], S);
array K[0..vsum]:=[nil, nil, ...];
begin
        K[0] := ∅; vmax := 0;
        for k := 1 to n do
        for v := vmax downto 0 do
            if K[v] ≠ nil then
            begin
                I := K[v];
                if Σ_{i∈I} s[i] + s[k] ≤ S then
                begin
                        v' := v + v[k]; I' := I ∪ {k};
                        if K[v'] = nil then K[v'] := I'
                        else begin
                                I'' := K[v'];
                                if Σ_{i∈I'} s[i] < Σ_{j∈I''} s[j] then
                                   K[v'] := I'
                        end;
                        if v' > vmax then vmax := v'
                end
            end;
        return (vmax, K[vmax])
end.
```

Algoritmi täyttää taulukkoa $K[0..vsum]$ $vsum = \sum_{i=1}^{n} v_i$, jonka alkioon $K[v]$ talletetaan paras kulloinkin löydetty, v:n arvoisen repun tuottava indeksijoukko. (Huom. taulukko saattaa olla hyvin harva, joten se voisi oikeastaan kannattaa toteuttaa v:n mukaan järjestettynä listana.)

Algoritmi voidaan toteuttaa niin, että se toimii ajassa $o(nv^*)$, missä v^* on optimaalinen repun arvo. Aikaraja on kuitenkin eksponentiaalinen, koska arvo v^* on eksponentiaalinen sen esityksen koon (bittiesityksen) suhteen. Tälläista algoritmia, jonka aikavaativuus on polynominen syötteessä esiintyvien lukuarvojen suhteen, sanotaan *pseudopolynomiseksi*.

Algoritmistä saadaan silti repunpakkausongelmalle täysin polynominen approksimointiskeema skaalaamalla ongelman tapauksessa esiintyviä lukuja sopivasti. (Samaa tekniikkaa voidaan itse asiassa soveltaa kaikkiin muihinkin pseudopolynomisiin algoritmeihin.)

Jaetaan kaikkien annetun repunpakkausongelman tapauksen tavaroiden arvot jollakin vakiolla c:

$$< s_1, \ldots, s_n, v_1, \ldots, v_n > \rightarrow < s_1, \ldots, s_n, \lfloor \frac{v_1}{c} \rfloor, \ldots, \lfloor \frac{v_n}{c} \rfloor, s >$$

Olkoon I alkuperäisen ja I_c skaalatun tapauksen optimiratkaisu. Tällöin:

$$\begin{aligned}
v(I) &= \sum_{i \in I} v_i \\
&\geq \sum_{i \in I_c} v_i && \text{[koska } I \text{ on alkup. tap. optimi]} \\
&\geq c \sum_{i \in I_c} \lfloor \frac{v_i}{c} \rfloor && \text{[lattia-aritmetiikan nojalla]} \\
&\geq c \sum_{i \in I} \lfloor \frac{v_i}{c} \rfloor && \text{[koska } I_c \text{ on skaalatun tap. optimi]} \\
&\geq c \sum_{i \in I} (\frac{v_i}{c} - 1) && \text{[lattia-aritmetiikaa]} \\
&= \sum_{i \in I} v_i - |I|c \\
&\geq v(I) - nc.
\end{aligned}$$

Siten on

$$v(I) \geq v(I_c) \geq v(I) - nc,$$

ja skaalatun tapauksen perusteella alkuperäiselle muodostettavan ratkaisun suhteellinen virhe on enintään

$$\frac{nc}{v^*} \leq \frac{nc}{v_{max}},$$

missä $v^* = v(I)$ ja $v_{max} = max\{v_i | i = 1, ..., n\}$.

Toisaalta skaalattu tapaus voidaan ratkaista edellä esitetyllä algoritmilla ajassa $O(nv^*/c) = O(n^2 v_{max}/c)$.

Valitsemalla annetulla $\epsilon > 0$ skaalausvakioksi

$$c = \frac{\epsilon \cdot v_{max}}{n}$$

saadaan siten ajassa $O(n^3/\epsilon)$ toimiva ϵ-approksimointialgoritmi repunpakkausongelmalle.

4.3 Luokan NP-rakenne ja muita vaativuusluokkia

Lause(R.Ladner 1975). Olkoon $A \notin P$ rekursiivinen kieli. Tällöin on olemassa kieli $D \in P$, jolla $A \cap D \notin P$ ja $A \cap D <_m^p A$ (so. $A \cap D \leq_m^p A, A \neq_m^p A \cap D$.).

Seuraus Jos $P \neq NP$, niin luokassa NP\P on kieliä, jotka eivät ole NP-täydellisiä.

Yhtään *luonnollista* esimerkkiä kielestä, joka olisi "NP-epätäydellinen" ehdolla $P \neq NP$, ei tunneta.

Tunnetuimmat esimerkit luokan NP ongelmista, joiden ei tiedetä olevan P:ssä eikä NP-täydellisiä, ovat:

Verkkoisomorfia: annettu suuntaamattomat verkot $G_1 = (V_1, E_1)$ ja $G_2 = (V_2, E_2)$. Voidaanko muodostaa bijektio $f : V_1 \to V_2$, jolla

$$(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2 \qquad \forall u, v \in V_1?$$

Yhdistetyt luvut (=ei-alkuluvut): Annettu luonnollinen luku n. Onko olemassa luonnolliset luvut k ja l, $1 < k, l < n$, joilla $n = k \cdot l$?

Yhdistetyt luvut-ongelman kohdalla voidaan osoittaa(V.Pratt 1975), että myös sen komplementtiongelma, so. alkulukujen tunnistaminen, kuuluu luokkaan NP. Tätä pidetään yleensä vahvana todisteena sen puolesta, että ongelma ei ole NP-täydellinen. (Itse asiassa monet arvelevat, että y.l.- ja alkulukuongelmat ovatkin P:ssä.)

Olkoon C jokin kieliluokka (erit. C=NP). Määritellään

co-C=$\{\bar{A} | A \in \mathcal{C}\}$.

Voidaan todeta, että co-P=P. Sen sijaan yleisesti ei uskota, että olisi co-NP=NP. (so, että NP olisi suljettu komplementoinnin suhteen).

Luokkaan NP $\cap$ co-NP kuuluvat ne luokan NP kielet (ongelmat), joiden komplementitkin ovat NP:ssä (esim. alkulukujen tunnistusongelma).

KUVA

Lause Kieli A on co-NP-täydellinen, jos ja vain jos sen komplementti $\bar{A}$on NP-täydellinen.

Tyypillinen co-NP-täydellinen ongelma on esim.

Tautologiat: Annettu lausekalkyylin kaava φ. Onko φ tautologia, so. toteutuuko se kaikilla muuttujien totuusarvoasetuksilla?

Määritellään edelleen tilavaativuusluokat:

$PSPACE = \{A | A$ voidaan tunnistaa polyn. työtilassa toimivalla alg.$\}$,
$NPSPACE = \{A | A$voidaan tunnistaa polyn. työtilassa toimivalla "epädet." alg.$\}$

On melko helppo osoittaa, että PSPACE=co-PSPACE, ja

$$P \subseteq NP,\ co-NP\ \subseteq\ PSPACE.$$ |Ei tiedetä, onko P=PSPACE.

Hämmästyttävää on, että tilavaativuusluokkien kohdalla "P=NP"-ongelma ei esiinny:

Lause (W. Sawitch 1973).

$$PSPACE = NPSPACE.$$

Tyypillisiä PSPACE-täydellisiä ongelmia ovat:

Kvantifioitujen Boolen kaavojen totuus. Annettu kaava

$\varphi = (Q_1 x_1)(Q_2 x_2)...(Q_n x_n)\psi(x_1, ..., x_n)$,

missä kukin Q_i on $\forall$ tai $\exists$ ja ψ on lausekalkyylin kaava, jonka ainoat muuttujat ovat $x_1, ..., x_n$. Onko φ tosi?

Säännöllisten lausekkeiden universaalisuus. Annettu aakkoston $\sum = \{0, 1\}$ säännöllinen lauseke r. (Sallitut operaatiot $\cup, \cdot, *$). Onko $L(r) = \sum^*$?

Tarkastellaan vielä eksponentiaalisia vaativuusluokkia:

$EXPTIME = \{A | A$ voidaan tunnistaa algoritmilla, joka toimii ajassa $2^{p(n)}$, p polynomi$\}$,
$EXPSPACE = \{A | A$ voidaan tunnistaa algoritmilla, joka toimii tilassa $2^{p(n)}$, p polynomi$\}$.

Lause:

$$PSPACE \subset EXPTIME \subset EXPSPACE$$

Jälleen minkään inkluusion aitoutta ei tiedetä varmasti; sen sijaan on melko helppo osoittaa:

Lause:

(i) P $\neq$ EXPTIME;

(ii) PSPACE $\neq$ EXPSPACE.

Seuraus: Jokin tai jotkin seuraavista sisältyvyyksistä ovat aitoja:

$$P \subset NP \subset PSPACE \subset EXPTIME.$$

Seuraus: Jos kieli A on EXPTIME- tai EXPSPACE-täydellinen, niin $A \notin P$.

Esim. ns. attribuuttikielioppien kehämäärittelyongelma on EXPTIME-täydellinen, säännöllisten lausekkeiden universaalisuusongelma operaatioilla $\{\cup, \cdot, *, \cap\}$ taas EXPSPACE-täydellinen.

Lause: (A. Meyer, L. Stockmeyer 1973). Säännöllisten lausekkeiden universaalisuusongelmaa operaatioilla $\cup, \cdot, *, \neg$ ei voida ratkaista ajassa $\underbrace{2^{2^{\cdots^{2^n}}}}_{k \ kpl}$.

Luku 5

Geometrisia algoritmeja

Algoritneja geometristen objektien (pisteet, janat, monitahokkaat, hypertasot,...) tehokkaaseen käsittelyyn.

Sovellusaloja: tietokonegrafiikka, robotiikka, VLSI, CAD, GIS, tilastotiede, hahmontunnistus,...

Tyypillinen ongelma: syötteenä annetaan joukko objekteja (pisteitä, janoja tms.); tuloksena halutaan jotain näiden geometrisia kombinaatioita (lähimmät naapurit, konveksi verho, leikkauspisteet,...)

Tyypillisissä sovelluksissa objekteja on yleensä *paljon* (10^4...10^7 ?), joten algoritmien tehokkuus on tärkeää (O($n\ log\ n$) ok, O(n^2) liikaa).

Seuraavassa esitetään joitakin tavallisimpia 2D-algoritmeja. Yleisillä d-ulotteisillakin olisi sovelluksia, mutta niiden vaativuus kasvaa usein valitettavasti kuten $\Omega(n^d)$

5.1 Peruskäsitteitä

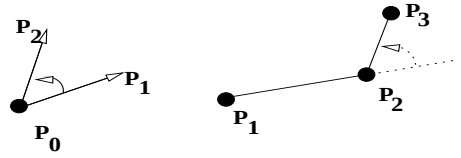
Pisteiden $p_1 = (x_1, y_1)$ ja $p_2 = (x_2, y_2)$ *konveksi kombinaatio* on mikä tahansa piste

$$p_3 = \lambda p_1 + (1 - \lambda)p_2, \quad 0 \leq \lambda \leq 1.$$

Näiden joukko on p_1:n ja p_2:n *yhdysjana* $\overline{p_1 p_2}$. (Joskus tarkastellaan myös *suunnattua yhdysjanaa* $p_1\vec{p}_2$, mutta seuraavassa puhutaan vain janoista.).

Algoritmisia perustehtäviä:

(i) Onko jana $\overline{p_0p_2}$ vasta- vai myötäpäivään janasta $\overline{p_0p_1}$, ts. kummalla on suurempi polaarikulma, kun origona on p_0



(ii) Onko käännös janalta $\overline{p_1p_2}$ janalle $\overline{p_2p_3}$ vasemmalle vai oikealle

(iii) Leikkaavatko janat $\overline{p_1p_2}$ ja $\overline{p_3p_4}$

Oleellista jatkon kannalta on, että nämä voidaan ratkaista tarvitsematta laskea minkäänlaisia kulma-arvoja.

Tason pisteiden $p_1 = (x_1, y_1)$ ja $p_2 = (x_2, y_2)$ (oikeastaan vektorien $(x_1, y_1, 0)$ ja $(x_2, y_2, 0)$ ristitulon kolmas komponentti) *ristitulo* on

$$p_1 \times p_2 = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} = -p_2 \times p_1$$

Tunnetusti:

$$p_1 \times p_2 \begin{cases} > 0, \text{ jos } p_2 \text{ on origosta katsoen vastapäivään pisteestä } p_1 \\ < 0, \text{ jos } p_2 \text{ on origosta katsoen myötäpäivään pisteestä } p_1 \\ = 0, \text{ jos } p_2 \text{ ja } p_1 \text{ ovat samalla origosuoralla} \end{cases}$$

Siten

(i) $\overline{p_0p_2}$ on vastapäivään $\overline{p_0p_1}$:stä, joss

$$(p_1-p_0)\times(p_2-p_0) = \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix} = (x_1-x_0)(y_2-y_0)-(x_2-x_0)(y_1-y_0) > 0.$$

(ii) käännös janalta $\overline{p_1p_2}$ janalle $\overline{p_2p_3}$ on vasemmalle, jos $(p_2 - p_1) \times (p_3 - p_2) > 0$

(iii) janat $\overline{p_1p_2}$ ja $\overline{p_3p_4}$ leikkaavat joss niiden "rajauslaatikot"ovat limittäin ja

$$((p_2 - p_1) \times (p_3 - p_1))((p_2 - p_1) \times (p_4 - p_1)) \leq 0,$$

$$((p_4 - p_3) \times (p_1 - p_3))((p_4 - p_3) \times (p_2 - p_3)) \leq 0$$

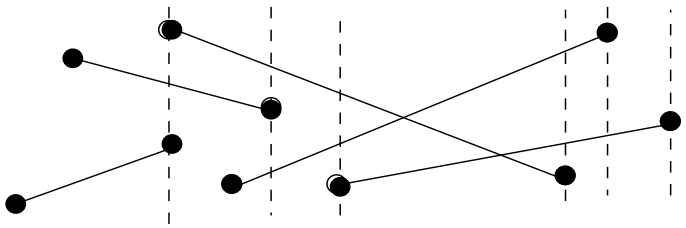
5.2 Janojen leikkauspisteet

Triviaalialgoritmi $\ominus(n^2)$
- Liikaa, jos leikkauspisteiden määrä k pieni
- Pahimmassa tapauksessa voi olla $k = \binom{n}{2}$.

Ns. "pyyhkäisymenetelmä"(sweap line technique) antaa algoritmin, jonka vaativuus on $O((n+k)log\ n)$.

Tarkastellaan ensin *yhden* leikkauspisteen hakua. Oletetaan yksinkertaisuuden vuoksi, että mikään jana ei ole pystysuora, ja että mitkään 3 janaa eivät leikkaa samassa pisteessä.

Algoritmi 1.: Janojen leikkauspiste. Pyyhkäisymenetelmä.
Syöte: Janat s_1,...,s_n (päätepiste-esitys)
Tulos: Leikkauspiste
Algoritmin idea:
- Järjestetään janojen päätepisteet ("tapahtumapisteet") x-koordinaatin suhteen nousevaan järjestykseen (Samalla x-koordinaaatilla pienin y-koordinaatti ensin)
- Kuljetetaan pystysuora "pyyhkäisyviiva"janojen yli vasemmalta oikealle, pitäen koko ajan kirjaa siitä mitkä janat leikkaavat viivan ja missä järjestyksessä y-koordinaatin suhteen.
- Huomataan, että vain naapurijanat (pyyhkäisyviivan suhteen) voivat leikata toisensa ja leikkauksessa niiden järjestys vaihtuu.



- Pyyhkäisyviivan esitys on järjestetty janajono T, johon voidaan kohdistaa seuraavat operaatiot
  - *insert(T,s)* - lisää (alkava) jana s kohdalleen T:hen
  - *delete(T,s)* - poista (päättyvä) jana s jonosta T
  - *above(T,s)* - palauta janan s yläpuolinen naapuri T:ssä
  - *below(T,s)* - palauta janan s alapuolinen naapuri T:ssä
- Toteutus esim. tasapainoisilla puilla $O(log\ n)$-operaatioin
- Järjestysrelaatio: jana s_1 on pienempi janaa s_2 abskissan arvolla x

ts. $s_1 <_x s_2$, jos
- x:n kautta kulkeva pystysuora leikkaa molempia

- leikkaa janan s_1 alempana kuin janan s_2
- Huom: janojen järjestys vaihtuu vain leikkauspisteissä

procedure *Etsi_leikkauspiste*(S) - S janajoukko
$T \leftarrow \emptyset$;
järjestä joukon S päätepisteet x-koordinaatin suhteen nousevaan järjestykseen (tarvittaessa pienin y-koordinaatti ensin)
for kaikille päätepisteille p järjestyksessä **do begin**
if p on janan s alkupiste **then begin**
insert(T,s);
if (jos jana *above(T,s)* olemassa ja leikkaa s:n) **or**
(jos jana *below(T,s)* olemassa ja leikkaa s:n)
then return leikkauspiste
end
if p on janan s päätepiste **then begin**
if molemmat janat *above(T,s)* ja *below(T,s)* ovat olemassa ja leikkaavat
then return leikkauspiste;
delete(T,s)
end
end
return ei leikkauspisteitä

Algoritmin aikavaativuus:
- päätepisteiden järjestäminen $O(n\ log\ n)$ askelta
- silmukka suoritetaan $2n$ kertaa, kukin suorituskerta $O(log\ n)$ askelta
- yhteensä $O(n\ log\ n)$ askelta

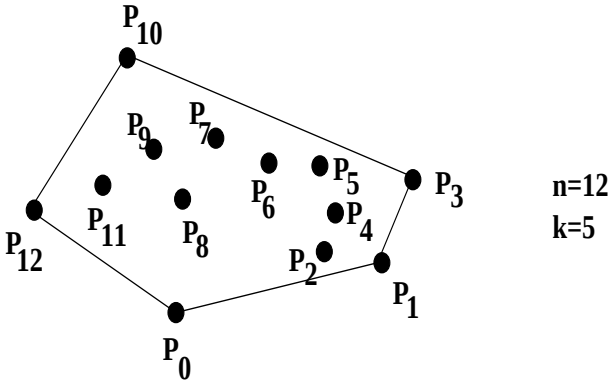
Algoritmi 2.: Janojen leikkauspisteet. Pyyhkäisymenetelmä.
Syöte: Janat s_1,...,s_n (päätepiste-esitys)
Tulos: Leikkauspisteet
Algoritmin idea:
- Janojen alku- ja loppupisteiden lisäksi myös leikkauspisteet ovat tapahtumapisteitä: näissä osallisten janojen järjestys vaihtuu.
→ Staattisen tapahtumajärjestyksen sijaan tapahtumalistaa täytyy ylläpitää prioriteettijonona. (Kekototeutus: $O(logn)$ askelta/operaatio.)
→ Aikavaativuus $O((n+k)log\ n)$.
- yksityiskohdat HT

5.3 Konveksi verho

Pistejoukon $Q = \{p_1, ..., p_n\}$ *konveksi verho* (convex hull) $CH(Q)$ on suppein konveksi monikulmio, joka sisältää kaikki Q:n pisteet. Laskennallisesti konveksin verhon määrääminen muodostuu niiden p_i-pisteiden löytämisestä,

joiden kautta piirretty monikulmio sisältää koko Q:n ja on konveksi, ts. kulmapisteiden yhdysjanat kulkevat monikulmion sisällä. Tavallisesti vaaditaan vielä, että algoritmin on annettava pisteet oikeassa kulmajärjestyksessä.

Esim:



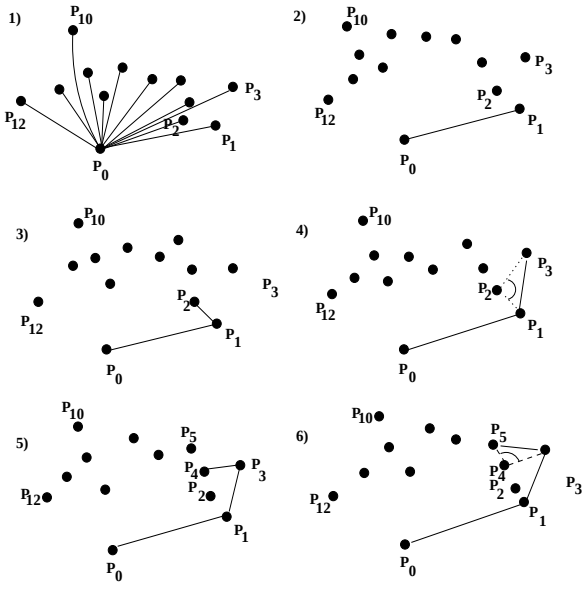
Useita vaihtoehtoisia tehokkaita algoritmeja:
- Grahamin "pyyhkäisymenetelmä"O($n\ log\ n$)
- Jarvisin "käärepaperialgoritmi"O($n{\cdot}k$)
- Inkrementaalinen menetelmä O($n\ log\ n$)
- Hajoita ja hallitse -menetelmä O($n\ log\ k$)

Algoritmi: Konveksi verho. Grahamin pyyhkäisymenetelmä.
Syöte: Joukon Q pisteet p_0,..., p_n
Tulos: Konveksin verhon $CH(Q)$ "kulmapisteet"q_1,..., q_k oikeassa järjestyksessä.
Algoritmin idea:
- Järjestetään pisteet nousevaan kulmajärjestykseen jostain referenssipisteestä (esim. alimmainen, vasemmanpuoleisin piste) lukien. Kulmien suuruuksien vertailua on selvitetty kohdassa 1. (Pisteet kannattaa ajatella skaalatun niin, että referenssipiste on origossa.)
- Käydään pisteitä läpi järjestyksessä ja kerätään joukkoon $CH(Q)$ kuuluvia ehdokaspisteitä pinoon S.
- Kun ehdokasverho tekee ei-konveksin käännöksen, ponnautetaan pinosta S pisteitä kunnes "mutka oikenee". (Peräkkäiset pisteet p, q ja r aiheuttavat ei-konveksin käännöksen, jos käännös janalta $\overline{pq}$ janalle $\overline{qr}$ on oikealle.)
- Proseduurin päättyessä pisteet $q_1, ..., q_k$ ovat pinossa S oikeassa järjestyksessä.

procedure *Graham_scan(Q)* - Oletetaaan $\mid Q \mid \geq 3$
Olkoon p_0 se Q:n piste, jolla on pienin y-koordinaatti, tai jos näitä on useita niin vasemmanpuoleisin
Järjestä Q:n muut pisteet nousevaan kulmajärjestykseen p_1,...,p_m
(jos samassa kulmassa on monta pistettä, säilytä vain etäisin)
empty_stack(S);
push(p_0,*S*);
push(p_1,*S*);
push(p_2,*S*);
for i:= 3 **to** m **do begin**
while pisteiden *next_to_top(S)*, *top(S)*, p_i muodostama käännös ei ole vasempaan **do** *pop(S)*;
push(p_i,*S*);
end

Aikavaativuus: Suoritusaika on vähintään $O(n\ log\ n)$, koska algoritmi sisältää pisteiden lajittelun. Itse kulmapisteiden määrääminen on kuitenkin $O(n)$-operaatio. Tämä seuraa siitä, että suoritus etenee aina uuteen pisteeseen p_i ja pinosta poistetaan vain joukon Q pisteitä (enintään $n-2$ poistoa).

Esim:

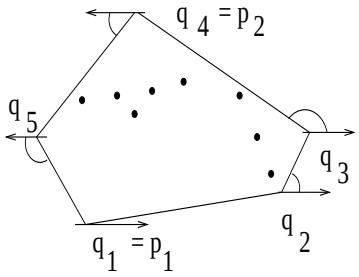


Algoritmi: Konveksi verho. Jarvisin käärepaperialgoritmi.
Syöte: Joukon Q pisteet p_1,..., p_n
Tulos: Konveksin verhon *CH(Q)* "kulmapisteet"q_1,..., q_k oikeassa järjestyksessä.
Algoritmin idea:
- Kiinnitetään muodostettava verho alimpaan pisteeseen q_1
- "Vedetään"se "tiukasti"seuraavaan pisteeseen q_2 (= pisteeseen, joka on q_1:sta katsoen pienimmässä kulmassa)
- Yleisesti pistettä q_i seuraava verhon $CH(Q)$ piste q_{i+1} on se, joka on q_i:stä katsoen pienimmässä kulmassa
- Ylimpään pisteeseen tultua voi olla kätevää laskea kulmat vastakkaiseen suuntaan, so. mod π.

procedure Jarvis_march(Q) - Oletetaan $|Q| \geq 2$.
Olkoon p_1 se Q:n piste, jolla on pienin y-koordinaatti (tai jos näitä on useita niin vasemmanpuoleisin), ja p_2 se Q:n piste, jolla on suurin y-koordinaatti (tai jos näitä on useita niin oikeanpuoleisin).
i:=1; q[i]:= p_1;
while q[i] $\neq p_2$ **do begin**
  etsi Q:n piste p, joka on q[i]:stä pienimmässä kulmassa positiivisen
    x-akselin suhteen
  i:= i+1; q[i]:= p; Q:=$Q \setminus \{p\}$
end
while q[i] $\neq p_1$ **do begin**
  etsi Q:n piste p, joka on q[i]:stä pienimmässä kulmassa negatiivisen
    x-akselin suhteen
  i:= i+1; q[i]:= p; Q:=$Q \setminus \{p\}$
end

Algoritmin aikavaativuus: O($k \cdot n$), missä $k =| CH(Q) |$. HT.

Esim:



5.4 Lähimmät pisteet

Triviaalialgoritmi O(n^2), missä n pisteiden lukumäärä.

Algoritmi: Lähimmät pisteet. Osittava algoritmi.
Syöte: Pisteet p_1,..., p_n
Tulos: Pistepari $p = (x_1, y_2)$ ja $q = (x_2, y_2)$, joiden euklidinen etäisyys

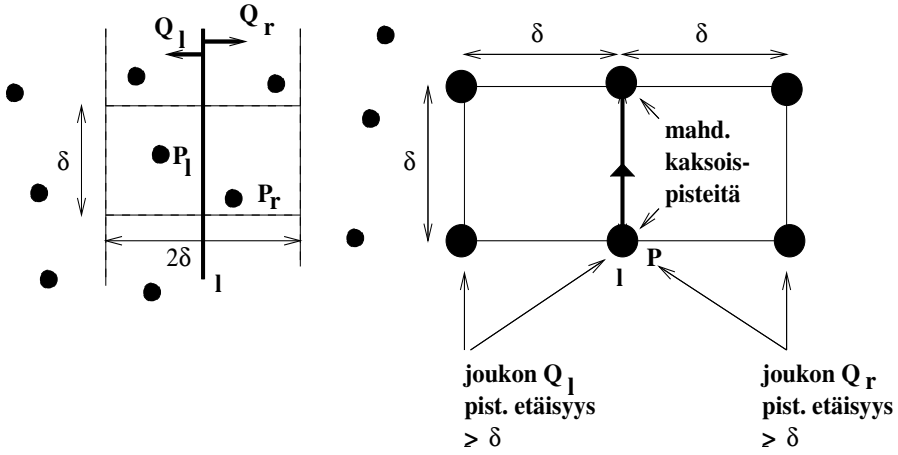
$$d(p,q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

on minimaalinen.
Algoritmin idea:
• *Alustus*
- Aluksi joukon Q pisteet järjestetään x-koordinaatin suhteen nousevaan järjestykseen joukkoon X ja y-koordinaatin suhteen nousevaan järjestykseen joukkoon Y. Saadaan ongelma $< Q, X, Y >$. (Tämä vaihe tarvitaan vain kerran; osaongelmien ratkaisussa lajitellut joukot tulevat suoraan rekursiolla)
• *Hajoita*
- Jos |Q|≤ 3, lähin piste määrätään suoraan; muuten:
- Määritetään joukon X järjestyksen perusteella jakosuora l (oletetaan että tällainen on olemassa), jonka vasemmalle ja oikealle puolelle sijoittuville pistejoukoille Q_L ja Q_R pätee:

$$\mid Q_L \mid = \lceil \mid Q \mid /2 \rceil \qquad \mid Q_R \mid = \lfloor \mid Q \mid /2 \rfloor$$

- Ositetaan joukot X ja Y järjestyksen säilyttäen Q-joukon jaon perusteella. Olkoot vastaavat joukot X_L, X_R, Y_L ja Y_R. Jaon perusteella esimerkiksi X_L on joukko Q_L järjestettynä x-koordinaatin suhteen.
- Määritetään rekursiivisilla kutsuilla $< Q_L, X_L, Y_L >$ ja $< Q_R, X_R, Y_R >$ lähimmät parit joukoista Q_L ja Q_R. Olkoot pienimmät etäisyydet vastaavasti δ_L ja δ_R, ja δ niiden minimi $\delta = min(\delta_L, \delta_R)$.
• *Hallitse*. Joukon Q lähin pari on nyt joko toinen jo löydetyistä (ehdokkaista), tai se sijaitsee jakosuoraa l ympäröivässä 2δ-levyisessä vyöhykkeessä. Tutkitaan vyöhyke seuraavasti.
- Muodostetaan joukosta Y järjestyksen säilyttäen joukko Y^δ, joka sisältää vain rajavyöhykkeen pisteet
- Käydään joukon Y^δ pisteet läpi y-koordinaatin mukaisessa järjestyksessä ja tutkitaan onko jokin seuraaja δ:aa lähempänä. Oleellista on, ettei tutkinnassa tarvitse läpikäydä koko Y^δ:n loppuosaa, vaan:

- *Tärkeä havainto*: Kunkin pisteen $p \in Y^\delta$ kohdalla riittää tutkia pelkästään 7 seuraajaa listasta Y^δ.



Aikavaativuus: Alustus $O(n\ log\ n)$ ja rekursioyhtälö

$$T(n) = 2T(n/2) + O(n)$$

jonka ratkaisusta $T(\mathrm{n}) = O(n\ log\ n)$.

5.5 Voronoi kaaviot (diagrammit)

Tärkeä geometrinen rakenne pisteiden läheisyysinformaation esittämiseen.

Lukuisia sovelluksia: lähimpien naapurien määrittäminen, lähipistekyselyihin vastaaminen, maksimaalisten tyhjien alueiden määrittäminen, Euklidiset virittävät puut,...

Määritelmä. Olkoon $d(p,q)$ pisteiden p ja q tavallinen Euklidinen etäisyys ja merkitään

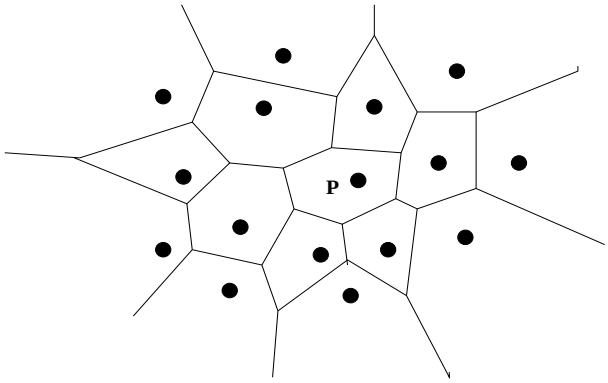
$$H(p,q) = \{r \in R^2 | d(p,r) \leq d(q,r)\}$$

Olkoon S äärellinen pistejoukko. Pisteen $p \in$ S *Voronoin solu* joukon S suhteen on

$$V_S(p) = \bigcap_{\substack{q \in S \\ q \neq p}} H(p,q)$$

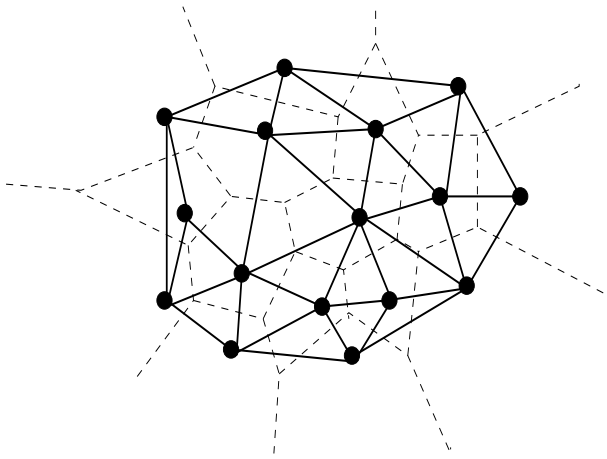
Voronoi solujen sivut ovat *Voronoi särmiä* ja näiden leikkauspisteet *Voronoin kärkiä*. Voronoin särmien ja kärkien joukko on S:n *Voronoin kaavio* $Vor(S)$.

Esim:



Määritelmä. Pistejoukon S Voronoin kaavion "duaali"$Del(S)$ on S:n *Delaunayn kolmiointi*, missä on särmillä yhdistetty toisiinsa täsmälleen ne S:n pisteet, joiden Voronoin solut ovat naapureita.

Esim:



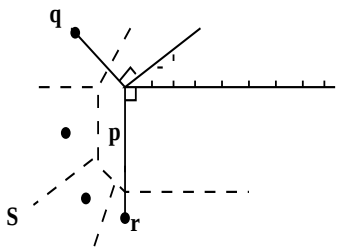
Tälläkin struktuurilla on paljon sovelluksia, esim funktion (pinnan) lineaarinen interpolointi joukossa S, FEM-menetelmän elementtihilojen muodostaminen.

Seuraavat lauseet luettelevat Voronoin kaavioiden perusominaisuuksia.

Lause 1. Pisteen p Voronoin solu $V(p)$ on konveksi monikulmio. Pisteiden p ja q, $p \neq q$, Voronoin soluilla on enintään yksi yhteinen särmä. Jos tällainen on, se on p:n ja q:n keskinormaalisuoran $e(p, q)$ osajana.

Lause 2. Voronoin solu $V(p)$ on rajoittamaton, jos ja vain jos piste p on joukon S konveksin verhon $CH(S)$ kärki.

Todistus. $\Rightarrow$ Oletetaan, että solu $V(p)$ on rajoittamaton, mutta $p \notin CH(S)$. Olkoon l jokin solun $V(p)$ sisältämä p:stä alkava puolisuora ja $\overline{qr}$ konveksin verhon reunajana, jonka tämä leikkaa. Nyt l:n pisteet riittävän kaukana p:stä ovat lähempänä q:ta tai r:ään kuin p:tä. Ristiriita.
$\Leftarrow$ Olkoon $p \in CH(S)$ ja q ja r sen naapurikärjet verhossa. Piirretään pisteeseen p janojen $\overline{qp}$ ja $\overline{pr}$ kanssa kohtisuorat puolisuorat. Näiden väliin jää rajoittamaton alue, jonka pisteet ovat lähempänä p:tä kuin muita S:n pisteitä.



Lause 3. Olkoon $\mid S \mid \geq 3$. Oletetaan, että mitkään neljä S:n pistettä eivät sijaitse samalla ympyräkehällä. Tällöin jokaisessa kaavion $Vor(S)$ kärjessä kohtaa toisensa tasan kolme särmää. Vastaavasti jokainen kaavion $Del(S)$ särmien rajaama alue on kolmio.

Lause 4. Olkoon $\mid S \mid \geq 3$. Kaavioissa $Vor(S)$ ja $Del(S)$ on tällöin kummassakin enintään $3 \mid S \mid -6$ särmää.

Todistus. Selvästi on $\mid Vor(S) \mid = \mid Del(S) \mid$. Kaavio $Del(S)$ on tasoverkko, jonka solmujoukko on S. Verkkoteoriasta tiedetään, että n solmun tasoverkossa voi olla enintään $3n-6$ kaarta. (Ns. Eulerin kaavan $\mid V \mid - \mid E \mid + \mid F \mid = 2$ (F=faces) seuraus).

Olkoon S tason pistejoukko, $\mid S \mid \geq 2$. Jaetaan S pystysuoralla jakoviivalla l osiin (oletetaan, ettei mikään piste sijaitse tällä viivalla)

$$S_L = \{(x,y) \in S \mid x < l_x\}, S_R = \{(x,y) \in S \mid x > l_x\},$$

missä l_x on viivan l x-koordinaatti. Merkitään lisäksi

$$P = \{p \in R^2 \mid d(p, S_L) = d(p, S_R)\},$$

missä $d(p,A) = min\{d(p,q) \mid q \in A\}$.

Lemma.

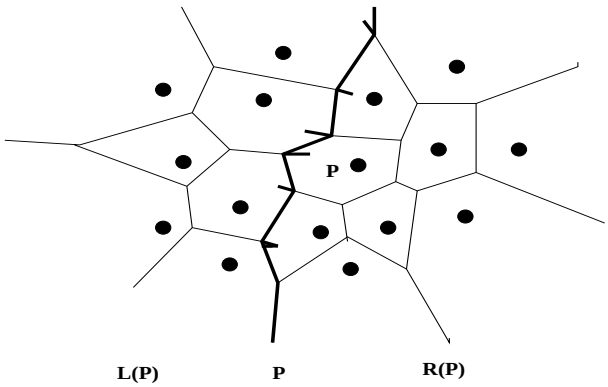
(i) $P = \{e(p,q) \bigcap Vor(S) \mid p \in S_L, q \in S_R\}$

(ii) $Vor(S) = (Vor(S_L) \bigcap L(P)) \bigcup P \bigcup (Vor(S_R) \bigcap R(P))$,

missä

$$L(P) = \{(x,y) \in R^2 \mid x < x' jollakin (x', y) \in P\}$$

$$R(P) = \{(x,y) \in R^2 \mid x > x' jollakin (x', y) \in P\}$$



Algoritmi: Voronoin kaavio - osittava algoritmi $VorCH(S)$.
Syöte: Tason joukko S
Tulos: Joukon Voronoin kaavio $Vor(S)$ ja konveksi verho $CH(S)$ parina $< Vor(S), CH(S) >$.
Algoritmin idea:
• *Alustus*
Aluksi joukon S pisteet järjestetään x-koordinaatin suhteen nousevaan järjestykseen. Tämä mahdollistaa nopean jakoviivojen määräämisen rekursion eri vaiheissa Tämä vaihe tarvitaan vain algoritmin alussa.
• *Hajoita*
Jos $S = \{p\}$, niin $Vor(S) = \emptyset$, $CH(S) = \{p\}$
Muuten, jaa S kahtia pystysuoralla jakoviivalla (oletetaan, että tämä voidaan tehdä). Olkoot osat S_L ja S_R. Kutsu rekursiivisesti

$< Vor_L, CH_L > := VorCH(S_L);$
$< Vor_R, CH_R > := VorCH(S_R);$

• *Hallitse*
Konveksien verhojen CH_L ja CH_R alapuolelle voidaan piirtää suora, jonka kulkee kummankin verhon jonkin kärjen kautta. Vastaavanlainen suora voidaan piirtää myös verhojen yläpuolelle. Suorilla olevia, kärkiä yhdistäviä janoja kutsutaan *ala-* ja *yläsilloiksi*. $CH(S)$ voidaan muodostaa verhoista CH_L ja CH_R unohtamalla tietyt reunat ja lisäämällä joukkoon sillat. Tässä ei puututa tarkemmin siltojen etsimiseen eikä $CH(S)$:n muodostamiseen.
Myös Voronoin kaavion $Vor(S)$ muodostamisessa käytetään konveksien verhojen ala- ja yläsiltoja. Aluksi on muodostettava edellä mainittu janajono P. Sen alin jana on alasillan keskinormaalin alkuosa (y-koordinaatin suhteen negatiivisesta äärettömyydestä) ja viimeinen jana on yläsillan keskinormaalin loppuosa (y-koordinaatin suhteen positiiviseen äärettömyyteen). Muut P:n janat ovat äärellisiä. Myös ne ovat S_L:n ja S_R:n pisteparien yhdysjanojen keskinormaalien osajanoja. *Tärkeä* havainto on, että pisteparin pisteet ovat aina osaverhojen pisteitä ja sijaitsevat verhoissa ala- ja yläsillan päätepisteiden välissä (verhojärjestyksessä). Janajonon muodostaminen aloitetaankin alasillan keskinormaalista ja lopetetaan yläsillan keskinormaaliin. Välillä olevat janat löytyvät tutkimalla CH_L:n pisteitä vastapäivään ja CH_R:n pisteitä myötäpäivään. Aloituspisteinä ovat alasillan päätepisteet $p \in CH_L$ ja $r \in CH_R$ sekä näiden verhoseuraajat q ja s. Pisteiden p ja r perusteella määräytyy alhaalta lähtevä puolisuora (P:n ensimmäinen jana). Puolisuoran päätepiste on alin piste, jossa se kohtaa joko Voronoin kaavion Vor_L tai Vor_R. Havainnon perusteella kaavioiden leikkaavat särmät ovat janojen $\overline{pq}$ ja $\overline{rs}$ keskinormaalien määräämiä. On siis määrättävä puolisuoran ja keskinormaalien leikkauspisteet ja valittava niistä alempi. Tästä pisteestä on taas lähdettävä ylöspäin. Suunnan määräämiseksi on sillä verholla, jonka pistepari (p ja q tai r ja s) määritteli alemman leikkauspisteen, edettävä.

Näin joko p:=q ja q:= q:n seuraaja tai r:=s ja s:= s:n seuraaja. Uusi suunta on nyt päivitetyn janan $\overline{pr}$ keskinormaalin mukainen. Näin jatketaan kunnes pistepariksi p, r tulee yläsillan päätepistepari. Sen jälkeen on vielä yläsillan loppuosa liitettävä P:hen.
Kun P on saatu muodostettua, tunnetaan kaikki $Vor(S)$:n särmät. Osa niistä saadaan suoraan kaavioista $Vor(S_L)$ ja $Vor(S_R)$. Muut ovat joko P:n osia tai pätkäistyjä osakaaavioiden särmiä. Pätkäistyt löytyvät itse asiassa samalla kun P:tä muodostetaan. Todellisuudessa siis kannattaakin myös koko $Vor(S)$:n muodostaminen suorittaa samalla, kun P haetaan.
Jotta kaikki onnistuisi, pitää Voronoin kaavio tallettaa kaksoislinkitettyyn listaan. Tällaisessa listassa on kukin särmä esitetty vain yhden kerran, joten sen tilatarve on lineaarinen. (Jos kaikki kaaavion monikulmiot talletettaisiin erillisinä, tarvittaisiin mahdollisesti neliöllinen tila.) Talletuksen yksityiskohtiin ei tässä puututa.

Algoritmin aikavaativuus. Edellä selvitetyin periaattein voidaan Voronoin diagrammi muodostaa ajassa $O(\mid S \mid log \mid S \mid)$.

procedure VorCH(S)
if $\mid S \mid = 1$**then return** $< \emptyset, S >$;
Osita S x-koordinaatin suunnassa yhtäsuuriin osiin S_L ja S_R;
Laske rekursiolla $< Vor_L, CH_L >$:= VorCH(S_L); $< Vor_R, CH_R >$:= VorCH(S_R);
P := tyhjä murtoviiva;
$\overline{pr}$:= verhojen CH_L, CH_R alasilta
l:= $e(p, r)$;{keskinormaali}
q := next(p,CH_L); {edetään verhossa vastapäivään}
s := $next(r,CH_R)$; {edetaan verhossa myötäpäivään}
repeat
t_L:= leikkauspiste($\overline{pq}, l$)
t_R:= leikkauspiste($\overline{rs}, l$)
if t_L on alempana kuin t_R **then**
begin
P:= P + (puoli)suora l ylöspäin edellisestä leikkauspisteestä leikkauspisteeseen t_L.
p:= q; q:= $next(q,CH_L)$
l:= $e(p, r)$
end
else
begin {symmetrinen tapaus}
P:= P + (puoli)suora l ylöspäin edellisestä leikkauspisteestä leikkauspisteeseen t_R.
r:= s; s:= next(s,CH_R)
l:= $e(p, r)$
end
until $\overline{rs}$ = yläsilta
P:= P + yläsillan keskinormaali viimeisestä leikkauspisteestä ylös;
CH := $CH_L \bigcup CH_R \bigcup$ {ylasilta,alasilta}\{siltojen väli}
return $< (Var(S_L) \bigcap L(P)) \bigcup P \bigcup (Var(S_R) \bigcap R(P)), CH >$

Luku 6

Merkkijonon haku

Probleemana on löytää annetusta merkkijonosta M[i], i= 1,...,n merkkijono P[i], i= 1,...,m. Perusalgoritmi tehtävän ratkaisemiseksi on

procedure rv_mjhaku(M,P)
pp:=1; mp:=1;
while pp $\leq$ m **and** mp $\leq$ n **do**
  if P[pp] = M[mp]
    then pp:= pp+1; mp:= mp+1;
    else pp:= 1; mp:= mp-pp+2;
if pp $>$ m
  then return mp-m;
  else return -1;
end

Algoritmi palauttaa arvon -1, jos merkkijonoa ei löydy, ja muuten (ensimmäisen) alkamiskohdan.

Algoritmin pahimman tapauksen suoritusaika on kertaluokkaa $\Theta(n \cdot m)$. Tämä nähdään esimerkiksi tarkastelemalla jonoja

M= $a^{n-1}b$
P= $a^{m-1}b$

Seuraavassa esitettävän *Knuth-Morris-Pratt*-menetelmän suoritusaika on sekä keskimäärin että pahimmassa tapauksessa kertaluokkaa $O(n+m)$

6.1 Knuth-Morris-Pratt-menetelmä

Tarkastellaan menetelmää esimerkin avulla. Olkoon

P= *abcabcacab*, m= 10.
M= *babcbabcabcaabcabcacabc*, n= 23,

missä alleviivaus ilmaisee tutkittavan M-jonon merkin. Koska verrattavat eivät ole samoja edetään tilanteeseen

P= *abcabcacab*
M= *babcbabcabcaabcabcacabc*.

Tällöin päällekäin olevat "nykyiset"alkiot ovat samoja, joten vain alleviivaus liikkuu. Näin jatketaan tilaan

P= *abcabcacab*
M= *babcbabcabcaabcabcacabc*.

Tässä vaiheessa törmätään erisuuriin alkioihin ja myös P-jonoa pitää siirtää. Kuinka paljon? Päätös voidaan perustaa sille, että tekstin lopussa ennen erisuuruutta ovat merkit

abc = P[1..3].

On siis löydettävä pisin sellainen P-jonon alku, joka täsmää merkkijonon

bc = P[2..3]

loppuun. Koska tällaista ei ole, voidaan suorittaa 3:n askeleen siirto. Huomattakoon, että päätös voidaan tehdä pelkästään P-jonon merkkien ja indeksikohdan pp = 3 avulla (nykyinen merkkikohta P-jonossa on pp+1).

Uusi tilanne on nyt

P= *abcabcacab*
M= *babcbabcabcaabcabcacabc*.

Eroavuuden vuoksi sekä P-jonoa että merkkikohtaa siirretään yhdellä. Etenemällä vielä muutama askel alleviivauksen siirtämisessä päädymme tilaan

P= *abcabcacab*
M= *babcbabcabcaabcabcacabc*.

Jälleen törmättiin erisuurteen ja P-jonon siirtämiseen. Nyt eroavuutta ennen ovat tekstimerkit

abcabca = P[1..7] (pp= 7).

Edellisen periaatteen mukaan on nyt löydettävä merkkijonon
bcabca = P[2..7]
lopusta pisin P-jonon alku. Tällainen on P[1..4]= *abca*, uusi pp= 4, siirto 3 = 7-4 ja tilanne

P= *abcabcacab*
M= *babcbabcabcaabcabcacabc*.

Erovuuden vuoksi on jälleen tehtävä P-jonon 3-askeleen siirto

P= *abcabcacab*
M= *babcbabcabcaabcabcacabc*.

Vielä yhden P-siirron ja muutaman alleviivaussiirron jälkeen merkkijono löytyy tilassa

P= *abcabcacab*
M= *babcbabcabcaabcabcacabc*.

Edellä olleessa oleeellista oli se, että P-jono saaattoi edetä usean askeleen ja alleviivaus ei koskaan siirry vasemmalle. Edelleen P-jonon jokainen siirto pystyttiin päättelemään kahdesta asiasta:

1) ei-täsmäys _-kohdassa

2) P-jonon nykyisen merkin kohta ts. tekstistä on löytynyt P[1..pp]

Näin ollen P-jonon liukuminen voidaan laskea etukäteen. Ilmoittakoon taulukkoarvo N[pp] (N = NEXT) mihin P-jonon alleviivauskohta (N[pp]+1) on asetettava, jos erisuuruus havaitaan P-jonon indeksissä pp+1. Eo. esimerkissä

pp=	1	2	3	4	5	6	7	8	9	10
N[pp]=	0	0	0	1	2	3	4	0	1	2

N-taulukkoa käyttävä haku voidaan suorittaa algoritmilla

```
procedure KMP_mjhaku(M,P,N)
pp:=0;
for mp = 1 to n do  begin
   while pp > 0 and M[mp] ≠ P[pp+1] do  pp:= N[pp];
   if M[mp] = P[pp+1] then pp:= pp + 1;
   if pp = m then return mp-m+1;
end
```

Ongelmana algoritmin suoritusajan määräämisessä on **while**-silmukan suorituskertojen laskeminen. Voidaan kuitenkin havaita, että jokainen suoritus-

kerta siirtää P-taulukkoa (N[t]<t). Niiden määrä ei tämän vuoksi voi olla n:ää suurempi. Koko algoritmin suoritusaika on näin ollen kertaluokkaa $O(n+m)$, koska N-taulukko voidaan laskea ajassa $O(m)$

Siirrytään nyt N-taulukon määräämiseen. Tilanteessa, jossa N-arvoa tarvitaan, on P-taulukon alla indekseissä 1,...,pp samat arvot. Näin N[pp] arvoksi pitää valita suurin sellainen i, että

(1) $0 \leq$ i $<$ pp
(2) P[1]...P[i] on jonon P[2]...P[pp] loppuosa

Ilmeisesti N[1]=0. Oletetaan, että N[1],...,N[s] on jo laskettu, ja tutkitaan miten N[s+1] saadaan määrätyksi. Ilmeisesti N[s+1] $\leq$ N[s]+1. Jos P[s+1]=P[N[s]+1], niin N[s+1]= N[s]+1. Muuten voidaan tilanne käsittää sellaiseksi, että merkkijonosta P[2..s+1] haetaan merkkijonoa P . Etsinnässä on löydetty jono P[1...N[s]], mutta tutkittavan jonon nykyinen merkki P[s+1] ja haettavan P-jonon nykyinen merkki P[t+1] (t=N[s]) ovat erisuuria. Tällöin on siis normaalilla tavalla siirrettävä P-jonoa oikealle. Koska t=N[s]<s, voidaan siirtäminen suorittaa jo lasketulla taulukolla N[1...s]. Jos vielä törmätään erisuuruuteen, voidaan siirto suorittaa samoin, jne. Laskenta voidaan siis tehdä seuraavasti

```
t:= N[s];
while t>0 and P[s+1] ≠ P[t+1] do t:= N[t];
if P[s+1] = P[t+1] then t:= t+1;
N[s+1]:= t;
```

Kaikkiaan saadaan N-laskennalle algoritmi

```
t:= 0; N[1]:=0;
for pp = 2 to m do begin
   while t > 0 and P[pp] ≠ P[t+1] do t:= N[t];
   if P[pp] = P[t+1] then t:= t+1;
   N[pp]:= t;
end
```

Algoritmin suoritusajan lineaarisuus voidaan perustella myös tasatulla analyysillä. Ongelmana ovat **while**-silmukan suorituskerrat. Valitaan potentiaaliksi t. Sen alkuarvo on nolla ja sitä kasvatetaan **for**-kierroksella korkeintaan kerran yhdellä. Koska se ei kuitenkaan koskaan muutu negatiiviseksi ja jokainen **while**-silmukan suoritus pienentää sitä (N[t]<t), voi suorituskertojen määrä olla korkeintaan m-1 (tai m-2, koska viimeinen t-lisäys **while**-silmukoiden jälkeen).

6.2 Boyer-Moore-Horspool-menetelmä

Knuth-Morris-Pratt-menetelmässä merkkijonon P siirtämistä nopeutettiin alkuosien esiintymisiä etukäteen tutkimalla. Toinen ajatus nopeuttamiselle on laskea etukäteen kunkin merkin esiintymät haettavassa merkkijonossa. Boyer-Moore-Horspool-menetelmässä tämä on toteutettu laskemalla aluksi kunkin merkin viimeinen esiintyminen P-merkkijonossa. Jos merkistönä on Σ, niin kirjaava taulukko LOR (Last OccuRrence) voidaan muodostaa seuraavasti

for $a \in \Sigma$ **do** LOR[a]= 0; /*Jos merkkiä ei löydy, on arvo 0*/
for pp = 1 **to** m **do** LOR[P[pp]]:= pp;

Alla oleva hakualgoritmi etenee P-merkkijonoa lopusta alkuun. Törmätessään eroavuuteen se siirtää P:tä niin että erisuuruuskohdalle tulee viimeinen P:n sopiva merkki. Jos P:stä ei löydy sopivaa, voidaan koko P vetää merkin ohi. Siihen on liitetty mahdollisuus tehostaa etsintää myös muilla tavoilla. Näin on tehtävä ainakin silloin kun menetelmä ehdottaa P:n siirtoa vasemmalle. Periaatteena menetelmässä on hyväksikäyttää tietoa kulloinkin tutkitun loppuosan sisällöstä. Vastaavasti kuin edellisessä menetelmässä voidaan tältä pohjalta tehdä siirtopäätöksiä pelkän P-jonon perusteella. Nyt vain on etsittävä P:n loppuosia (suffix) ei alkuosia (prefix) kuten N-taulukkoa määrättäessä. Tässä ei esitetä menetelmän yksityiskohtia, mutta mainittakoon että menetelmä hyväksikäyttää käännetyn merkkijonon P^R N-taulukkoa.

```
procedure BMH_mjhaku(M,P)
s= 0;
while s≤n-m do
   pp= m;
   while pp>0 and P[pp]= M[s+pp] do pp= pp-1;
   if pp=0 then return s
         else s= s+max(pp-LOR[s+pp],);
```

Harjoitustehtäviä

1. Määritä rekursioyhtälön

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T(n/4) + \sqrt{n}, \qquad \text{kun } n = 4^k, \quad k \geq 1. \end{aligned}$$

   tarkka ratkaisu, kun n on neljän potenssi.

2. Eräissä "hajota ja hallitse"-tyyppisissä algoritmeissa voidaan n:n kokoinen ongelman tapaus osittaa tasaisesti $\sqrt{n}$:n kokoisiin osatapauksiin niin, että algoritmin aikavaativuutta $T(n)$ tulee kuvaamaan rekursiokaava:

$$T(n) \leq \begin{cases} c & \text{kun } n = 2 \\ \sqrt{n}T(\sqrt{n}) + bn, & \text{kun } n > 2. \end{cases}$$

   Määrää O-yläraja funktiolle $T(n)$. (*Vihje:* Tee muuttujanvaihto $n = 2^k$.)

3. Ratkaise generoivia funktioita käyttäen rekursioyhtälö

$$\begin{cases} t_0 = 0, \quad t_1 = 1, \\ t_n = 5t_{n-1} - 6t_{n-2}, \quad n \geq 2. \end{cases}$$

   (*Vihje:* Kun olet määrittänyt jonon $\langle t_n \rangle$ generoivan funktion, yritä kirjoittaa sen nimittäjänä oleva lauseke muotoon $(1-az)(1-bz)$ ja sovella sitten osamurtolukuhajotelmaa.)

4. Tiedetään (Anal. pk/jk), että nollan ympäristössä suppeneva potenssisarja voidaan derivoida termeittäin, so. jos $G(z) = \sum_{k\geq 0} p_k z^k$, niin $G'(z) = \sum_{k\geq 1} k p_k z^{k-1}$.

   (a) Johda tämän tiedon avulla jonon $\langle 1, 2, 3, 4, \ldots \rangle$ generoiva funktio jonon $\langle 1, 1, 1, 1, \ldots \rangle$ generoivasta funktiosta.

   (b) Oletetaan, että jonon $\langle p_k \rangle$ arvot ovat jonkin diskreetin todennäköisyysjakauman pistetodennäköisyyksiä, siis esim. "p_k = todennäköisyys, että algoritmin suoritus vaatii tasan k askelta". Mitä on tällöin $G(1)$? Entä mikä on arvon $G'(1)$ tulkinta?

5. Ohjelma käsittelee kahta pinoa S ja T, joihin voidaan tavallisten pino-operaatioiden (*push, pop*) lisäksi kohdistaa kopiointi-operaatioita *copy*(S,T) ja *copy*(T,S). Suunnittele pinoille taulukkototeutus, jossa kopiointi-operaation yhteydessä ainoastaan edellisen kopioinnin jälkeen tapahtuneet muutokset päivitetään pinosta toiseen. Osoita, että tässä toteutuksessa minkä tahansa aluksi samansisältöisiin pinoihin kohdistuvan n operaation jonon kokonaissuoritusaika on vain $O(n)$.

6. (a) Esitä binäärisen hakupuun päivitysalgoritmien toiminta, kun aluksi tyhjään puuhun ensin lisätään alkiot 6, 2, 7, 0, 4, 5, 9, 8, ja näin saadusta puusta sitten poistetaan alkio 2.

   (b) Osoita, että n alkion lisääminen aluksi tyhjään binääriseen hakupuuhun vaatii pahimmassa tapauksessa $\Omega(n^2)$ vertailuoperaatiota.

7. Osoita, että vertailujen määrän odotusarvo, kun aluksi tyhjään binääriseen hakupuuhun lisätään n satunnaista alkiota, on $O(n \log n)$. (*Ohje:* Olkoot lisättävät alkiot $a_1, \ldots, a_n$. Puun juureen sijoittuva alkio a_1 on todennäköisyydellä $\frac{1}{n}$ joukon k:nneksi pienin. Tässä tapauksessa vasempaan alipuuhun sijoittuu $k-1$ alkiota ja oikeaan alipuuhun $n-k$ alkiota. Muodosta tältä pohjalta vertailujen määrän odotusarvoa kuvaava rekursioyhtälö ja ratkaise se joko arvauksen sovitusmenetelmällä tai ns. historian eliminoinnilla (ks. pikalajittelualgoritmin analyysi, muistiinpanojen s. 35).)

8. (a) Esitä (2,3)-puun päivitysalgoritmien toiminta, kun aluksi tyhjään puuhun ensin lisätään alkiot 6, 2, 7, 0, 4, 5, 9, 8, ja näin saadusta puusta sitten poistetaan alkio 0.

   (b) Osoita, että n alkion lisääminen aluksi tyhjään (2,3)-puuhun vaatii enintään $O(n \log n)$ operaatiota.

9. (a) Esitä AVL-puun lisäysalgoritmin toiminta, kun aluksi tyhjään puuhun lisätään alkiot 6, 2, 7, 0, 4, 5, 9, 8.

   (b) Suunnittele AVL-puille poistoalgoritmi ja esitä algoritmisi toiminta, kun edellisessä tehtävässä muodostetusta puusta poistetaan alkio 2.

10. Osoita, että AVL-puussa, jonka korkeus on h, on oltava vähintään $\Omega(\phi^h)$ solmua, missä ϕ on ns. kultaisen leikkauksen suhde, $\phi = (1 + \sqrt{5})/2$. (*Ohje:* Johda puun rakenteen perusteella rekursioyhtälö pienimmän h:n korkuisen puun solmujen määrälle ja ratkaise se.) Päättele tämän tiedon perusteella, että n-solmuisen AVL-puun korkeus on luokkaa $O(\log n)$.

11. Osoita, että universaalihajautusta käytettäessä minkä tahansa avaimen x kanssa törmäävien muiden avainten $y \neq x$ määrän odotusarvo on alle 1 (vrt. Cormen/Leiserson/Rivest, *Introduction to Algorithms*, ss. 230–231). Päättele tästä, että hakemisto-operaatiot MEMBER, INSERT ja DELETE voidaan universaalihajautusta käyttäen toteuttaa odotusarvoisesti vakioajassa. (*Huom.* Odotusarvo lasketaan siis hajautinten, ei syötteenä saatavien avainkokoelmien suhteen. Kaikki avainkokoelmat ovat samassa asemassa.)

12. (a) Suunnittele *splay*-puurakenteelle tehokkaat, alkion juureennosto-operaatioon perustuvat päivitysalgoritmit. (*Ohje:* Kunkin operaation aluksi nostetaan käsiteltävä alkio puun juureen.)

    (b) Esitä *splay*-puun päivitysalgoritmien toiminta, kun aluksi tyhjään puuhun ensin lisätään alkiot 6, 2, 7, 0, 4, 5, 9, 8, ja näin saadusta puusta sitten poistetaan alkiot 2 ja 9.

13. Täydennä luennolla (muistiinpanojen ss. 164.12–13) esitetty *splay*-puiden tasatun vaativuuden analyysi osoittamalla, että myös alkion x siksikkierron tasattu kustannus on enintään $3(R'(x) - R(x))$, missä $R(x)$ on alkion x ranki ennen kiertoa ja $R'(x)$ sen ranki kierroksen jälkeen.

14. Simuloi kekotietorakenteen *heapify*-operaation (luennot, s. 166) toimintaa sen muodostaessa lukujonoa 4, 11, 9, 10, 5, 6, 8, 1, 2, 16 vastaavaa kekoa. (Pienin alkio keon juureen.)

15. Muodosta joukkoja {5, 2, 7} ja {0, 3, 4, 6, 1, 8, 9} vastaavat binomimetsät ja simuloi niiden yhdistämistä luennolla (muistiinpanojen ss. 167–169) esitetyillä algoritmeilla.

16. Todista seuraavat binomipuiden rakenneominaisuudet:

    (a) Binomipuun B_n, $n \geq 1$, juuren alipuut ovat täsmälleen binomipuut $B_0, \ldots, B_{n-1}$.

    (b) Binomipuussa B_n, $n \geq 0$, on syvyydellä k, $0 \leq k \leq n$, täsmälleen $\binom{n}{k}$ solmua

17. Millainen puurakenne syntyy, kun luennolla (ss. 172–173) esitettyä tasapainottavaa ja tiivistävää *union-find*-algoritmia sovelletaan seuraavan ohjelman tuottamaan *union-find*-operaatiojonoon? Aluksi on $S_i = \{i\}, i = 1, \ldots, 16$.

```
begin
    for i = 1 to 15 by 2 do union(i, i + 1, i);
    for i = 1 to 13 by 4 do union(i, i + 2, i);
    union(1, 5, 1);
```

union(9, 13, 9);
union(1, 9, 1);
for $i = 1$ **to** 13 **by** 4 **do** *find*(i)
end.

18. Osoita, että jos *union-find*-algoritmissa käytetään puiden tasapainotusta, mutta ei polun tiivistystä, n:n *union-find*-operaation jonon suorittaminen voi vaatia ajan $\Omega(n \log n)$. (*Vihje:* Tarkastele jonoja, joista syntyy edellisen tehtävän tapaan binomipuita.)

19. Tarkastellaan muotoa

$$X_i = X_j, \quad i, j = 1, 2, \ldots, k,$$

olevien yhtälöryhmien ratkaisemista, missä kukin literaali X_i on muuttuja tai kokonaisluku. Esimerkiksi ryhmällä

$$\begin{aligned} X &= 3 \\ X &= Y \end{aligned}$$

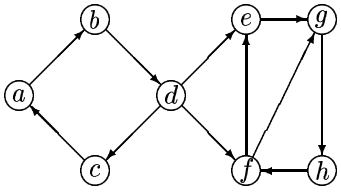
on ratkaisu $X = Y = 3$, mutta laajennetulla ryhmällä

$$\begin{aligned} X &= 3 \\ X &= Y \\ Y &= 4 \end{aligned}$$

ei ole ratkaisua. Selitä, miten *union-find*-algoritmilla voidaan tehokkaasti päättää, onko annetulla yhtälöryhmällä ratkaisua vai ei.

20. Todista seuraava väite (luentojen s. 180, Lause 2): Jos (v, w) on syvyyshaun määrittämä suunnatun verkon sivuttaiskaari, niin *postnum*[v] > *postnum*[w].

21. Määritä alla olevan verkon vahvasti yhtenäiset komponentit luentojen s. 181 algoritmilla. Oletetaan, että solmut ovat vieruslistoissa aakkosjärjestyksessä ja että algoritmi aloittaa verkon tutkimisen solmusta a.



22. Sanotaan, että suunnatun verkon $G = (V, E)$ solmu u on *alkusolmu*, jos verkon kaikki muut solmut voidaan saavuttaa u:sta, so. jos kuhunkin

muuhun solmuun johtaa u:sta suunnattu polku. Suunnittele tehokas algoritmi, joka etsii vieruslistoina esitetyn suunnatun verkon jonkin alkusolmun, mikäli verkossa sellaisia on. Miten muuttaisit algoritmia, jos tehtävänä olisi määrittää verkon *kaikki* alkusolmut?

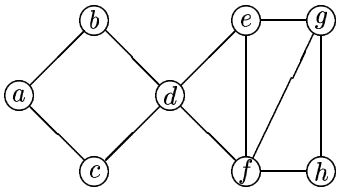
23. Todista seuraavat yhtenäisen suuntaamattoman verkon G artikulaatiopisteitä ja 2-yhtenäisiä komponentteja koskevat väitteet (luentomuistiinpanojen s. 186):

    (a) Kaaret e_1 ja e_2 kuuluvat G:n samaan 2-yhtenäiseen komponenttiin, jos ja vain jos $e_1 = e_2$ tai G:ssä on sykli joka sisältää molemmat kaaret.

    (b) Kahdella G:n 2-yhtenäisellä komponentilla on enintään yksi yhteinen solmu.

    (c) G:n artikulaatiopisteet ovat täsmälleen sen 2-yhtenäisten komponenttien leikkaussolmut.

24. Määritä alla olevan verkon 2-yhtenäiset komponentit luennolla esitetyllä algoritmilla (muistiinpanojen s. 189a). Oletetaan, että solmut ovat vieruslistoissa aakkosjärjestyksessä ja että algoritmi aloittaa verkon tutkimisen solmusta a.



25. Laadi algoritmi, joka tutkii onko syötteenä annettu suuntaamaton verkko kaksijakoinen. Algoritmin tulee toimia verkon kaarien määrän suhteen lineaarisessa ajassa. (Syöteverkon solmuista ei siis ole valmiiksi sanottu, mitkä ovat jaon "vasemmalla" ja mitkä "oikealla" puolella. Huomaa kuitenkin, että kaksijakoisuusehdon mukaan "vasemman puolen" solmuista voi olla kaaria vain "oikean puolen" solmuihin ja päinvastoin.)

26. *Binäärinen De Bruijnin jono* on 2^k-bittinen binäärijono $a_1 a_2 \ldots a_{2^k}$, joka syklisesti luettaessa sisältää kaikki k-bittiset binäärijonot osajonoinaan. (Esimerkiksi 00010111 on eräs De Bruijnin jono, missä $k = 3$.) Luonnostele algoritmi, joka tuottaa n-bittisen De Bruijnin jonon, missä n on muotoa 2^k, n:n suhteen lineaarisessa ajassa. (*Vihje:* Tarkastele suunnattua verkkoa G_k, jonka solmut vastaavat kaikkia mahdollisia $k-1$-bittisiä binäärijonoja, ja solmusta $b_1 b_2 \ldots b_{k-1}$ on symboleilla 0 ja 1 nimetyt suunnatut kaaret solmuihin $b_2 \ldots b_{k-1}0$ ja $b_2 \ldots b_{k-1}1$.)

27. Tarkastellaan seuraavaa "edustajien valinta"-ongelmaa. Syötteenä annetaan kokoelma perusjoukon $U = \{1, \ldots, n\}$ osajoukkoja $S_1, \ldots, S_k$. Tehtävänä on valita kustakin joukosta alkio $x_i \in S_i$ siten, että $x_i \neq x_j$, kun $i \neq j$. Hahmottele algoritmi edustajien valintaan. (*Vihje:* Kaksijakoisten verkkojen pariutus.)

28. Tarkastellaan kauppamatkustajan ongelmaa täydellisessä verkossa G, jonka solmuina ovat tason pisteet (i, j), missä $i = 1, 2, 3$ ja $j = 1, 2, 3$. Solmuja on siis yhdeksän kappaletta. Solmujen välisten kaarten kustannukset ovat samat kuin vastaavien pisteiden euklidiset etäisyydet. Simuloi luennolla (muistiinpanojen ss. 143–146) esitettyjen kahden ΔTSP-approksimointialgoritmin ("kahdesti puun ympäri" -tekniikka ja Christofidesin algoritmi) toimintaa tässä verkossa. Kokeile kahta tai useampaa virittävää puuta algoritmien lähtökohtana. Laske myös algoritmien tuottamien ratkaisujen suhteelliset virheet.

29. Tarkastellaan seuraavaa solmupeiteongelman ahnetta approksimointialgoritmia: Verkosta valitaan peitteeseen lisättäväksi solmu, jonka asteluku on mahdollisimman suuri, ja poistetaan kaikki siihen liittyvät kaaret. Tätä toistetaan, kunnes verkossa ei ole enää kaaria jäljellä. Osoita, että algoritmi voi pahimmassa tapauksessa valita peitteen C, jolla $|C| > 2|C^*|$, missä C^* on optimipeite.

30. Ratkaise repunpakkausongelma, jossa tavaroiden "koot" ovat (4, 1, 2, 3, 2, 1, 2) ja "arvot" vastaavasti (299, 73, 159, 221, 137, 89, 157) soveltamalla luennolla (ss. 149–152) esitettyä ϵ-approksimointiskeemaa parametrin ϵ arvolla 1. Repun maksimikoko on 10.

31. Tarkastellaan seuraavaa NP-täydellistä *lokerointiongelmaa* (engl. Bin Packing):

    > Annettu joukko välin $(0, 1)$ reaalilukuja $x_1, \ldots, x_n$. Luvut on ositettava mahdollisimman pieneen määrään osajoukkoja ("lokeroita") siten, että kunkin osajoukon alkioiden summa on korkeintaan 1.

    Ongelman ahne *first fit*-approksimointialgoritmi toimii seuraavasti: Luvut käsitellään järjestyksessä $x_1, \ldots, x_n$. Luku x_i sijoitetaan lokeroiden järjestyksessä ensimmäiseen lokeroon, johon se mahtuu. Osoita, että tämä menettely on 1-approksimointialgoritmi, so. tarvitsee enintään kaksinkertaisen määrän lokeroita optimaaliseen nähden. (*Vihje:* Osoita, että *first fit*-heuristiikka täyttää kaikki lokerot, paitsi mahdollisesti yhden, ainakin puolilleen.)

32. Todista luennolla esitetty Seurauslause 7.1 (muistiinpanojen s. 153): Jos P $\neq$ NP, niin luokassa NP \ P on kieliä (ongelmia), jotka eivät ole NP-täydellisiä.

33. (a) Todista seuraava luennolla esitetyn Lauseen 7.8 (muistiinpanojen s. 154) yleistys: Olkoon $\mathcal{C}$ jokin kieliluokka (so. ongelmaluokka, esim. $\mathcal{C}$ = P, NP, PSPACE,...). Kieli (ongelma) A on co-$\mathcal{C}$-täydellinen, jos ja vain jos sen komplementtikieli $\bar{A}$ on $\mathcal{C}$-täydellinen.

    (b) Osoita tämän tuloksen perusteella, että lausekalkyylin kaavojen tautologisuusongelma (muistiinpanojen s. 154) on co-NP-täydellinen.

34. Sanotaan, että Monte Carlo -tyyppinen satunnaisalgoritmi A on *luotettava*, jos se annetulla syötteellä antaa oikean vastauksen todennäköisyydellä $p > 0$, ja todennäköisyydellä $q = 1 - p$ vastaa "en tiedä". Mikä on oikean vastauksen todennäköisyys, kun algoritmi A suoritetaan k kertaa peräkkäin? (Oletetaan, että suorituskerrat ovat toisistaan riippumattomia, ja toisto lopetetaan heti kun oikea vastaus on saatu.) Montako kertaa algoritmia tarvitsee enintään toistaa, jotta oikean vastauksen todennäköisyys saataisiin suuremmaksi kuin $1 - \epsilon$, jollakin annetulla $\epsilon > 0$?

35. Muodostetaan edellisen tehtävän algoritmin A pohjalta Las Vegas -tyyppinen algoritmi B, joka toistaa A:ta, kunnes saa oikean vastauksen (so. A vastaa jotakin muuta kuin "en tiedä"). Mikä on tarvittavien toistojen määrän odotusarvo?

36. Olkoon A jonkin päätösongelman ratkaiseva (so. 0/1-arvoinen) Monte Carlo -algoritmi, joka antaa oikean vastauksen todennäköisyydellä $p > 1/2$ ja väärän vastauksen todennäköisyydellä $q = 1 - p$. (Tällä kertaa algoritmi ei siis vastaa luotettavasti "en tiedä", vaan saattaa suorastaan valehdella.) Muodostetaan algoritmista A algoritmi B, joka toistaa A:ta $k = 2t - 1$ kertaa ja valitsee vastaukseksi sen, jonka A antoi useammin, so. vähintään t kertaa. Arvioi todennäköisyyttä, että B antaa väärän vastauksen, kun A:n eri suorituskerrat ovat toisistaan riippumattomia. Erityisesti: monellako A:n toistolla saadaan väärän vastauksen todennäköisyys pienemmäksi kuin jokin annettu $\epsilon > 0$? (Käytä luentomuistiinpanojen s. 209 esitettyjä Chernoffin rajoja.)

37. Todista seuraavat hyppylistojen ominaisuudet:

    (a) n alkion hyppylistan alkioiden maksimikorkeus on "suurella todennäköisyydellä" enintään $2 \log_2 n$.

    (b) Kahden korkeutta h olevan alkion välissä on odotusarvoisesti $O(1)$ korkeutta $h - 1$ olevaa alkiota. (*Vihje:* Huomaa, että jos alkion

x korkeus on vähintään $h-1$, niin todennäköisyydellä $1/2$ sen korkeus on itse asiassa h tai suurempi.)

38. Lausekalkyylin kaava φ on *disjunktiivisessa normaalimuodossa (dnf)*, jos se on muotoa

$$\varphi = \varphi_1 \vee \ldots \vee \varphi_m,$$

missä kukin termi φ_i on literaalien konjunktio

$$\varphi_i = \tilde{x}_{i1} \wedge \ldots \wedge \tilde{x}_{ir_i}.$$

(Tässä siis kukin $\tilde{x}_{ij}$ on joko muuttuja tai sellaisen negaatio.) Laadi satunnaisalgoritmi, joka arvioi monellako muuttujien totuusarvoasetuksella annettu dnf-muotoinen kaava φ tulee todeksi. (*Idea:* Kukin kaavan termi φ_i määrittelee tietyn toteuttavien totuusarvoasetusten joukon S_i, ja koko kaavan φ toteuttavien asetusten joukko on näiden S_i-joukkojen yhdiste. Sovella luentomuistiinpanojen s. 211 esitettyä joukkoyhdisteen kokoa arvioivaa satunnaisalgoritmia.) *Lisätieto:* Voidaan osoittaa, että jos P $\neq$ NP, niin toteuttavien asetusten määrää ei voida laskea tarkasti polynomisessa ajassa.

39. Osoita, että jos $p > 2$ on alkuluku, niin kaikilla a, $1 \leq a < p$, on voimassa $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$. (Vrt. luentomuistiinpanojen s. 222.) Käytä hyväksesi seuraavia algebrallisia perustietoja:

    (a) Jos p on alkuluku, niin nollasta poikkeavien kokonaislukujen mod p muodostama ryhmä $\mathcal{Z}_p^* = (\mathcal{Z}_p \setminus \{0\}, \cdot)$ on syklinen, so. jollakin alkiolla $g \in \mathcal{Z}_p^*$ on $\mathcal{Z}_p^* = \{g^k \pmod{p} \mid k \geq 0\}$.

    (b) Kaikilla a, $1 \leq a < p$, on $a^{p-1} \equiv 1 \pmod{p}$ (ns. Fermat'n pieni lause).

40. Testaa luennolla (s. 225) esitetyllä Solovayn–Strassenin algoritmilla ainakin 75% varmuudella, onko 103 alkuluku. (*Huom:* Luentomuistiinpanoissa on valitettavasti virhe s. 224 kohdassa (d): Gaussin resiprookkilause t. "neliönjäännösten käännösyhtälö" pätee vain, kun Jacobin symbolissa $\left(\frac{y}{x}\right)$ luvut y ja x ovat parittomia ja keskenään jaottomia. Jos symbolin "osoittaja" y on parillinen, voidaan sen sijaan soveltaa symbolin multiplikatiivisuuteen perustuvaa sääntöä $\left(\frac{2z}{x}\right) = \left(\frac{2}{x}\right)\left(\frac{z}{x}\right)$.)

41. Johda luentomuistiinpanojen s. 233 esitetty, simuloidun jäähdytyksen asymptoottista konvergenssia kun $T \to 0$ koskeva Seurauslause 2 samalla sivulla esitetystä, algoritmin käyttäytymistä vakiolämpötilassa $T > 0$ koskevasta Lauseesta 1.

42. Tarkastellaan seuraavaa yksinkertaista "säämallia". Säätila annettuna päivänä voi olla joko aurinkoinen (a), pilvinen (p) tai sateinen (s). Seuraavan päivän säätilan todennäköisyysjakauma riippuu edellisen päivän säätilasta seuraavan siirtymämatriisin esittämällä tavalla (rivi-indeksi kuvaa päivän t, sarakeindeksi päivän $t+1$ säätilaa):

	a	p	s
a	0.5	0.5	0
p	0.5	0.25	0.25
s	0	0.5	0.5

Totea, että matriisin määrittelemällä "sääprosessilla" on tasapainojakauma, ja määritä se.

43. Hahmottele simuloitua jäähdytystä käyttävä ratkaisumenetelmä NP-täydelliselle solmupeiteongelmalle (engl. VERTEX COVER; luentomuistiinpanojen s. 117). Valitse sopiva kustannusfunktio ja naapurustorakenne. Varmista, että osaat tuottaa annetun kelvollisen ratkaisun satunnaisia naapuriratkaisuja tasaisen jakauman mukaan. Minkälaista jäähdytysaikataulua ongelman ratkaisemiseen n solmun verkossa tulisi käyttää luennolla esitetyn konvergenssilauseen 3 (s. 244) mukaan?

44. Osoita, suoraan koordinaattiesityksiä tarkastelemalla, että jos $p_1 = (x_1, y_1)$ ja $p_2 = (x_2, y_2)$ ovat tason pisteitä, niin $p_1 \times p_2 = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} > 0$, jos ja vain jos piste p_2 on origosta katsoen vastapäivään pisteestä p_1.

45. Tehtävänä on laatia algoritmi, joka tutkii muodostaako annettu pistejono $p_1, p_2, \ldots, p_n$ järjestyksessä konveksin monikulmion kulmapisteet. Yksi idea olisi tarkastaa, että pistejonon määrittämän murtoviivan käännökset ovat joko kaikki vasempaan tai kaikki oikeaan. Mikä vika tässä ideassa on? Anna oikea, pistejonon pituuden suhteen lineaarisessa ajassa toimiva algoritmi.

46. Suunnittele tehokas algoritmi sen tutkimiseen, menevätkö kaksi kulmapistejonoina esitettyä yksinkertaista monikulmiota päällekkäin. (Monikulmio on *yksinkertainen*, jos se on yhtenäinen eivätkä sen sivut leikkaa toisiaan.)

47. Todista, että n-solmuisessa tasoverkossa (siis esimerkiksi n pisteen Delaunayn kolmioinnissa) voi olla enintään $3n - 6$ kaarta, kun $n \geq 3$. (*Ohje:* Todista ensin Eulerin kaava, jonka mukaan n-solmuisessa yhtenäisessä tasoverkossa, jossa on m kaarta ja p tahkoa (engl. faces), on voimassa yhtälö $n - m + p = 2$. [Huomaa, että verkon "ulkopuoli" lasketaan myös tahkoksi.] Tämän kaavan todistus on induktio m:n

suhteen: kullakin m:n arvolla tarkastellaan ensin erikseen tapaus, jossa verkko on syklitön, so. puu, ja sitten palautetaan syklin sisältävä verkko yksinkertaisempaan poistamalla syklistä yksi kaari. Haluttu tulos seuraa Eulerin kaavasta, kun ensin arvioidaan kaarien määrälle alaraja tahkojen määrän funktiona.)

48. Olkoon annettuna tason pistejoukko S. Suunnittele tehokas algoritmi, joka sijoittaa tasoon uuden pisteen siten, että se sijoittuu S:n konveksin verhon sisään, mutta on maksimaalisen kaukana kaikista S:n pisteistä. (*Vihje:* Sovella Voronoin kaavioita.)