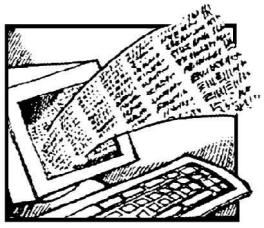


Mitä ovat laskennalliset tieteet? (Symbolinen laskenta)

Jukka Tuomela
Fysiikan ja matematiikan laitos, Itä-Suomen yliopisto
jukka.tuomela@uef.fi

Aluksi

Määritelmän mukaan π on ympyrän kehän suhde halkaisijaan. Miten sille voitaisiin laskea hyvä likiarvo? Miten polynomin nollakohdat voisi laskea? Miten Google järjestää haun tulokset paremmuusjärjestykseen? Mitä reittiä pitkin raketti saadaan Marsiin mahdollisimman vähällä polttoaineella? Monet ongelmat sekä puhtaassa matematiikassa että reaalimaailmassa johtavat mielenkiintoisiin laskennallisiin tehtäviin. Lisäksi tietotekniikan leviäminen lähes kaikille elämän alueille takaa sen, että uusia laskennallisia tehtäviä tulee koko ajan lisää.

Tarkastelen seuraavassa laskennallisuutta lähinnä matematiikan kannalta: mitä uusia haasteita tai näkökulmia laskennalliset tehtävät antavat matematiikalle ja matemaatikoille. Heikkola ja Tarvainen [4] puolestaan äskettäin käsittelivät laskennallisuutta sovellusten ja mallinnuksen lähtökohdista käsin, joten kirjoitukset täydentävät sopivasti toisiaan.

Laskenta ja (tiede)politiikka

Laskennallisista tieteistä on puhuttu ja kirjoitettu viime aikoina varsin paljon. Opetusministeriö julkaisi jokin aika sitten paksun raportin laskennallisista tieteistä, perustettiin laskennallisten tieteiden seura (SULATIS) ja Suomen Akatemian rahoilla on käynnistetty sekä laskennallisten tieteiden tutkimusohjelma että laskennallisten tieteiden tohtoriohjelma (FICS).[1] On annettu ymmärtää, että laskennallisilla tieteillä olisi laajempaakin yhteiskunnallista merkitystä. Erityisesti on korostettu laskennallisten tieteitten merkitystä korkean teknologian tuotekehityksen kannalta. Jätän kuitenkin näitten laajempien kuvioitten pohtimisen lukijalle poliittiseksi harjoitustehtäväksi ja keskityn vain laskennallisten tieteitten matemaattisempaan puoleen.

Mitä laskennallisilla tieteillä oikeastaan tarkoitetaan? Ehkäpä seuraava määritelmä antaa jonkinlaisen kuvan asiasta:

laskennallisissa tieteissä tutkitaan jonkin sovellusalueen ilmiötä käyttäen matemaattisia malleja, joitten käsittely ja ratkaisu edellyttää mittavia tietokoneresursseja.

Tarvitaan siis kolmenlaista tietämystä: itse sovellusalueen, matematiikan ja tietojenkäsittelyn. Tuskinpa kukaan yksittäinen tutkija hallitsee kaikkia kolmea osaaluetta, joten luontevinta on harjoittaa tutkimusta ryh-

[1] http://www.minedu.fi/OPM/Julkaisut/2007/Laskennallisen_tieteen_kehittaminen_Suomessa.html
http://www.sulatis.fi/
http://www.aka.fi/fi/A/Tiedeyhteiskunnassa/Tutkimusohjelmat/kaynnissa/Laskennalliset-tieteet/
http://fics.hiit.fi/

missä, joihin kuuluu eri alojen asiantuntijoita.

Esimerkiksi sään ennustaminen täyttää nämä tunnusmerkit: meteorologian ja fysiikan perusteella päädytään tiettyyn malliin (osittaisdifferentiaaliyhtälösysteemiin), jonka numeerinen ratkaiseminen vie varsin ison siivun kaikesta Suomessa käytössä olevasta superkonekapasiteetista. Tämäntyyppistä lähestymistapaa voidaan soveltaa moniin muihinkin ongelmiin, ja edellä mainitusta opetusministeriön raportista löytyy runsaasti esimerkkejä.

Laskennallisten menetelmien tutkimus on aktiivista sekä Suomessa että muualla. Sen sijaan laskennallisten tieteitten vaikutus matematiikan opetukseen on toistaiseksi ollut varsin vaatimatonta. Ehkäpä eräs syy on laskennallisten tieteitten monitieteisyys: tutkimuksessa tieteitten raja-aitoja on helpompi ylittää kuin opetuksessa. Miten sitten laskennallinen näkökulma pitäisi ottaa huomioon vaikkapa kouluopetuksessa onkin sitten jo vaikea koulutuspoliittinen kysymys, eikä siihen tässä voi puuttua.

Katsotaan seuraavassa muutamien esimerkkien avulla, minkälaisiin kysymyksiin laskennallinen näkökulma johtaa. Numeeriseen laskentaan liittyvät asiat jätetään kirjoituksen seuraavaan osaan.

Algoritmi

Keskeinen käsite kaikessa laskennassa on algoritmi. Lyhyesti algoritmi voitaisiin määritellä seuraavasti:

ALG *algoritmi on äärellinen joukko ohjeita, joiden perusteella tehdään äärellinen määrä toimenpiteitä, jotka tuottavat halutun lopputuloksen*

Kun on annettu jokin algoritmi, pitää siis osoittaa, että

KR1 *algoritmi päättyy* ja

KR2 *algoritmi antaa oikean tuloksen.*

Tyypillisesti algoritmit toteutetaan tietokoneen avulla. Tällöin ohjelmalle annetaan alussa joitain syötteitä (input). Sitten ohjelmaa ajetaan äärellinen aika (KR1), minkä jälkeen saadaan oikea vastaus (KR2). Vaikka yllä olevat kriteerit vaikuttavat itsestään selviltä, niin jokainen joka on ohjelmoinut, on tehnyt ohjelmia, joille KR1 ja/tai KR2 **EI** päde!

Geometria

Vaikka algoritmi tuntuukin ehkä varsin modernilta käsitteeltä, niin sillä on hyvin pitkä historia. Itse asiassa jo Eukleideen geometria oli algoritmista.[2] Eukleideen kirjassa on kahdenlaisia tuloksia: lauseita ja konstruktio-ongelmia. Jälkimmäisissä käsitellään nimenomaan algoritmeja. Muistetaan, että Eukleideen geometriassa voidaan tehdä kahdenlaisia toimenpiteitä:

OP1 *kahden pisteen kautta voidaan piirtää suora* (viivoitin käytössä)

OP2 *kun keskipiste on annettu, voidaan piirtää ympyrä* (harppi käytössä)

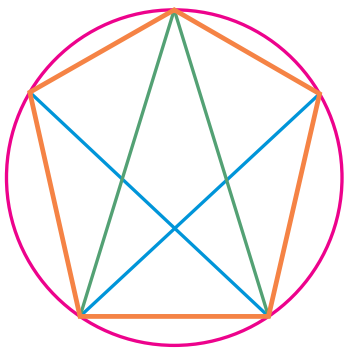
Kaikki konstruktiot voidaan siten palauttaa äärelliseksi määräksi operaatioita OP1 ja OP2. Tarkastellaan esimerkiksi Eukleideen IV kirjan ongelmaa 11.

ONGELMA: Piirrä annetun ympyrän sisälle tasasivuinen viisikulmio.

Eukleides antaa seuraavan algoritmin, kuva 1.

ALGORITMI IV 11

1. piirrä tasakylkinen kolmio, jolle kärkikulma on puolet kahdesta muusta kulmasta (algoritmi IV 10)[3]
2. piirrä samanmuotoinen kolmio annettuun ympyrään (algoritmi IV 2)
3. puolita kolmion isommat kulmat (algoritmi I 9)
4. yhdistä pisteet (OP1)



Kuva 1: Algoritmin IV 11 toteutus.

Kuva on piirretty vain suurin piirtein oikeaksi, joten lopputulos ei ole täsmälleen säännöllinen viisikulmio.

[2] Netistä löytyy useitakin versioita Eukleideen geometriasta:
http://aleph0.clarku.edu/~djoyce/java/elements/toc.html
http://sunsite.ubc.ca/DigitalMathArchive/Euclid/byrne.html
http://farside.ph.utexas.edu/euclid/
Ne, joille kelpaa vain alkukielinen teksti:
http://www.physics.ntua.gr/~mourmouras/euclid/

[3] Eukleides kutsuu aliohjelmaa, joka on annettu IV kirjan ongelmassa 10.

Tähän liittyen muistetaan Poincarén tunnettu kommentti: *Geometria on taito tehdä hyviä päätelmiä huonosti piirretyistä kuvioista.*[4] Selvästi algoritmi päättyy, koska lopulta jokainen aliohjelma voidaan suorittaa tekemällä äärellinen määrä operaatioita OP1 ja OP2. Annettuaan algoritminsa Eukleides käyttää sivun osoittaakseen, että KR2 on voimassa.

Polynomit

Edellä algoritmin päättyminen oli itsestään selvää. Näin ei kuitenkaan aina ole. Katsotaan seuraavassa erästä algoritmia, jossa algoritmin päättymisen osoittamisen idea on erittäin tärkeä monissa yhteyksissä.

Olkoon $f = a_0 + a_1 x + \cdots + a_q x^q$ jokin q-asteinen polynomi ja merkitään $\mathsf{deg}(f) = q$. Olkoon edelleen g jokin toinen polynomi. Haluttaisiin laskea polynomien f ja g suurin yhteinen tekijä. Määritelmän mukaan $h = \mathsf{syt}(f, g)$, jos

(S1) $f = hf_1$ ja $g = hg_1$, missä f_1 ja g_1 ovat joitain polynomeja,

(S2) h on korkea-asteisin polynomi, jolla on tämä ominaisuus.

Tämä määritelmä ei ole ollenkaan algoritminen siinä mielessä, että se ei suoraan anna mitään ideaa siitä, miten h voitaisiin laskea. Eräs mahdollisuus perustuu polynomin jakamiseen tekijöihin. Tunnetusti q-asteisella polynomilla on q (mahdollisesti kompleksista) nollakohtaa, kun kertaluvut otetaan huomioon. Toisin sanoen voidaan kirjoittaa

$$f = a_q(x - c_1)^{k_1} \ldots (x - c_\ell)^{k_\ell},$$

missä $k_1 + \cdots + k_\ell = q$. Vastaavasti g voidaan jakaa tekijöihin ja tällöin suurin yhteinen tekijä saadaan katsomalla, kuinka paljon f:llä ja g:llä on yhteisiä nollakohtia. Tässä on kuitenkin se huono puoli, että pitäisi erikseen laskea nollakohdat, ja herääkin kysymys, onko tämä todella tarpeen. Itse asiassa on olemassa paljon tehokkaampi menetelmä. Olkoon annettu polynomit f ja g. Tällöin pätee, että on olemassa polynomit s (osamäärä) ja r (jakojäännös) siten, että

$$f = s\,g + r, \quad \mathsf{deg}(r) < \mathsf{deg}(g). \tag{1}$$

Lukija voi helposti edelleen osoittaa, että saadut s ja r ovat yksikäsitteisiä.[5] Oletetaan, että on käytössä aliohjelma jakojäännös, joka laskee r:n, kun f ja g on annettu. Tällöin seuraava algoritmi laskee f:n ja g:n suurimman yhteisen tekijän.

ALGORITMI SYT

```
h := f ; p := g
WHILE p ≠ 0 DO
    r:=jakojäännös(h,p);
    h:=p;
    p:=r;
ENDWHILE
```

Algoritmin päättyessä $p = 0$ ja $h = \mathsf{syt}(f, g)$. Algoritmin oikeellisuuden todistaminen jätetään harjoitustehtäväksi. Algoritmin päättyminen seuraa siitä, että se tuottaa jonon jakojäännöksiä r_ℓ siten, että

$$\mathsf{deg}(g) > \mathsf{deg}(r_1) > \cdots > \mathsf{deg}(r_\ell) > \cdots$$

Koska $\mathsf{deg}(p) \geq 0$ kaikille polynomeille ($p \neq 0$), niin yllä oleva jono ei voi jatkua mielivaltaisen pitkään. Pitää siis olla olemassa jokin $k \leq \mathsf{deg}(g)$ siten, että $r_k = 0$.

Tämä on hyvin tyypillistä. Usein algoritmeissa on jokin silmukka, jota toistetaan, kunnes jokin ehto tulee voimaan. Eräs todistustekniikka on osoittaa, että joka kierroksella jokin positiivinen, kokonaislukuarvoinen suure pienenee. Koska tätä ei voi tapahtua äärettömän monta kertaa, silmukka voidaan suorittaa vain äärellisen monta kertaa. Huomattakoon, että tässä saatiin lisäksi yläraja kierrosten lukumäärälle: $\mathsf{deg}(g)$. Tästä puolestaan saadaan luonnollisesti arvio, kuinka kauan algoritmin suoritus voi korkeintaan kestää.

Katsotaan vielä eräs sovellus syt-algoritmista. Olkoon f annettu kuten yhtälössä (1). Yksinkertaisin polynomi, jolla on samat nollakohdat, on siis

$$f_r = (x - c_1) \ldots (x - c_\ell).$$

Nyt f_r saadaan kaavasta

$$f_r = \frac{f}{\mathsf{syt}(f, f')},$$

missä f' on f:n derivaatta. Tässäkään ei siis tarvinnut laskea nollakohtia, mikä kenties tuntuu yllättävältä. Erinomainen johdatus laskennalliseen polynomialgebraan on [3].

Lajittelu ja laskennan vaativuus

Kaikessa käytännön laskennassa oleellista on myös kolmas kriteeri:

KR3 *algoritmi on riittävän nopea.*

[4] Géométrie est l'art de bien raisonner sur des figures mal faites.

[5] Vihje: käytä funktion deg ominaisuuksia.

Tietenkin tätä voidaan pitää myös kriteerin KR1 tarkennuksena: ei vaadita pelkästään, että algoritmi päättyy, vaan halutaan arvioida mahdollisimman tarkasti, kuinka kauan algoritmin suoritus kestää. Tällaista algoritmien tehokkuuden tutkimista kutsutaan *laskennan vaativuusanalyysiksi*. Hyödylliseksi on osoittautunut ajatella, että algoritmilla on käytössä tiettyjä alkeisoperaatioita, ja vaativuusanalyysissä pyritään arvioimaan, kuinka monta operaatiota algoritmin suorittaminen vaatii. Eukleideen geometrian tapauksessa voitaisiin laskea, kuinka monta alkeisoperaatiota OP1 ja OP2 tarvitaan tietyn algoritmin suorittamiseksi. Numeerisessa laskennassa taas lasketaan liukulukuoperaatioiden määrä.[6]

Algoritmin tehokkuuden analyysissä on kätevää ottaa käyttöön eräs hyödyllinen merkintä. Olkoot f ja g joitain funktioita. Merkintä $f(n) = O(g(n))$ tarkoittaa, että on olemassa jotkin vakiot a ja c siten, että

$$|f(n)| \leq c\, g(n) \quad \text{kun} \quad n > a.$$

Jatkossa ajatellaan, että n on kokonaisluku, mutta määritelmää käytetään myös, kun n on reaaliluku. Edelleen sanotaan, että f kasvaa polynomiaalisesti, jos $f(n) = O(n^k)$ jollekin k.

Laskennan vaativuusanalyysissä tyypillisesti tarkastellaan tilannetta, kun tehtävän koko kasvaa. Useinhan samalle algoritmille voidaan antaa erikokoisia syötteitä. Jos siis ajatellaan, että n kuvaa tehtävän kokoa ja $f(n)$ työmäärää, kun ratkaistaan kokoa n oleva tehtävä, niin haluttaisiin saada selville, miten nopeasti f kasvaa, kun $n \to \infty$.

Toisaalta säännöllisen viisikulmion tapauksessa algoritmilla ei oikeastaan ole syötteitä, mutta voidaan silti kysyä, olisiko olemassa jokin algoritmi, joka vaatisi vähemmän operaatioita OP1 ja OP2 kuin Eukleideen antama algoritmi. Tämän pohtimisen jätän harjoitustehtäväksi.

Tarkastellaan seuraavassa lajittelun vaativuusanalyysiä: olkoon annettu n kappaletta kokonaislukuja; kuinka nopeasti ne voidaan laittaa suuruusjärjestykseen? Luonnollisesti yhtä hyvin voitaisiin lajitella vaikkapa nimiä aakkosjärjestykseen, mutta tarkastellaan nyt (yleisyyttä rajoittamatta) kokonaislukuja. Lajittelussa on luonnollista ottaa alkeisoperaatioksi kahden luvun vertailu. Ongelma on siis laskea, kuinka monta vertailua tarvitaan, kun n lukua laitetaan suuruusjärjestykseen.

Lajittelussa jonkin algoritmin löytäminen on varsin helppoa, kukapa ei olisi joskus laittanut lukuja suuruusjärjestykseen, nimiä aakkosjärjestykseen tai vaikkapa ihmisiä pituusjärjestykseen. Koko ongelman mielenkiinto onkin nimenomaan vaativuusanalyysissä: miten lajittelun voisi tehdä mahdollisimman tehokkaasti.

Olkoon annettu

$$9 \quad 7 \quad 5 \quad 11 \quad 8 \quad 20 \quad 3 \quad 17.$$

Eräs algoritmi tunnetaan nimellä *kuplalajittelu* (bubble sort)[7]. Lähdetään liikkeelle vasemmalta ja etsitään suurinta arvoa. Ensin todetaan, että $9 > 7$, joten vaihdetaan niitten paikkaa. Sitten $9 > 5$, joten näittenkin paikkaa vaihdetaan. Näin jatketaan, ja 7 vertailun ja muutaman paikanvaihdon jälkeen saadaan

$$7 \quad 5 \quad 9 \quad 8 \quad 11 \quad 3 \quad 17 \quad 20.$$

Nyt tiedetään, että 20 on suurin, joten seuraavalla kierroksella pitää tehdä vain 6 vertailua. Lopulta 7 kierroksen jälkeen saadaan

$$3 \quad 5 \quad 7 \quad 8 \quad 9 \quad 11 \quad 17 \quad 20.$$

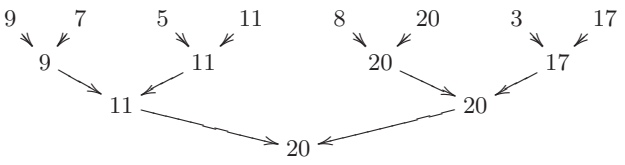
Yleisessä tapauksessa joudutaan siis tekemään

$$1 + 2 + \cdots + n - 1 = \tfrac{1}{2}\, n(n-1)$$

vertailua, joten työmäärä on $O(n^2)$. Huomattakoon, että paikanvaihtojen lukumäärä riippuu annetusta datasta, mutta vertailujen määrä on aina sama.

Tämä on varsin hidasta, jos n on hyvin iso. Tässä muuten lukijalle kenties tulee mieleen, että koska tietokoneet tulevat koko ajan yhä nopeammiksi, niin tällainen vaativuusanalyysi ei ehkä ole niin kovin tärkeää. Ehkäpä hieman paradoksaalisesti asia on päinvastoin: kun tietokoneet tulevat tehokkaammiksi, niin samalla heti halutaan käsitellä yhä suurempia datamääriä ja yhä monimutkaisempia malleja. Tämä on mahdollista vain, jos algoritmit ovat tarpeeksi tehokkaita.

Voitaisiinko siis lajitella nopeammin? Tarkastellaan erästä *kekolajittelun* muunnelmaa (heap sort). Käytetään samaa dataa kuin yllä ja tehdään siitä ”tenniskaavio”:

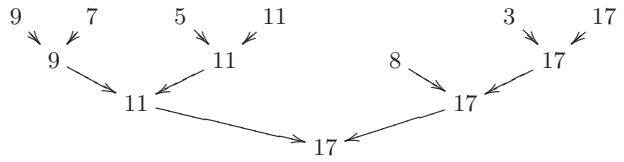


20 on ”voittaja”. Poistetaan se ja järjestetään jäljelle jäävät luvut samalla tavalla kaavioon. Huomaa, että läheskään kaikkia ”otteluja” ei tarvitse pelata uudelleen: nyt 8 siirtyy suoraan toiselle kierrokselle, ja sitten

[6] Liukulukuoperaatio (flop, floating point operation) on tietokoneella liukuluvuilla suoritettu yhteen-, vähennys-, kerto- tai jakolasku.

[7] Nimitys tulee kuulemma siitä, että jos luvut järjestettäisiin pystysuoraan, niin suuremmat luvut nousevat alhaalta ylös kuten ilmakuplat vedessä.

suoritetaan vertailut $8 < 17$ ja $11 < 17$. Saadaan siis



Uudelleenjärjestelyssä tarvittiin vain 3 operaatiota, koska kaavion "syvyys" on $3 = \log_2 8$.

Samaan tapaan jatkamalla ja poistamalla aina kaavion voittaja saadaan vastaus. Mutta onko tämä todella nopeampi yleisessä tapauksessa kuin kuplalajittelu? Oletetaan yksinkertaisuuden vuoksi, että $n = 2^k$. Alkuperäisen kaavion tekemiseen tarvitaan

$$1 + 2 + \cdots + 2^{k-1} = 2^k - 1 = n - 1$$

vertailua. Koska kaavion syvyys on $\log_2 n = k$, niin kaavion uudelleenjärjestelyssä joudutaan tekemään korkeintaan k operaatiota. Järjestely joudutaan tekemään n kertaa, joten kokonaistyömäärä on $O(n \log n)$. Isoilla n:n arvoilla kekolajittelu on siis huomattavasti nopeampi kuin kuplalajittelu. Voidaan lisäksi osoittaa, että itse asiassa vaativuus $O(n \log n)$ on optimaalinen: minkä tahansa lajittelumenetelmän työmäärä kasvaa vähintään tätä vauhtia [7].

Lopuksi

Edelliset esimerkit toivottavasti valaisivat hiukan eräitä puolia laskennallisista kysymyksistä matematiikan kannalta. Seuraavassa muutamia muita ideoita ja viitteitä.

Mikä on alkeisoperaatio?

Teorian "voima" kasvaa, jos voidaan ottaa lisää alkeisoperaatioita käyttöön. Esimerkiksi Eukleideen geometriaa voidaan laajentaa ottamalla käyttöön eräitä lisäoperaatioita. Voidaan vaikkapa olettaa, että viivoitin on merkitty, jolloin pituuksia voi mitata. Tämän avulla voidaan ratkaista kaksi klassista ongelmaa, jotka ovat mahdottomia "puhtaassa" euklidisessa geometriassa: kuution kahdentaminen ja kulman jakaminen kolmeen osaan [2, 8]. Samalla tavalla kvanttitietokone tarjoaa periaatteessa uusia mahdollisuuksia, koska se voi suorittaa alkeisoperaatioita, jotka tavalliselle tietokoneelle ovat mahdottomia [1]. Tosin kvanttitietokoneen käytännön toteutus tuntuu olevan vielä pitkän työn takana.

Vaativuus

Algoritmien vaativuuden tarkastelussa pitää ottaa myös huomioon muistitilan tarve. Esimerkiksi polynomilaskennassa tämä on varsin vakava ongelma. Tyypillisessä tilanteessa polynomin kertoimet ovat joko rationaalilukuja tai "parametreja", siis oikeastaan myös muuttujia, mutta koska niiden luonne tehtävässä on erilainen, niin niitä myös sitten käsitellään eri tavalla. Joka tapauksessa eräissä tärkeissä algoritmeissa, joista puhutaan tarkemmin kirjassa [3], ongelmana on, että polynomien kertoimien lausekkeet saattavat kasvaa hyvinkin suuriksi. Samoin polynomien asteluku saattaa kasvaa voimakkaasti. Edelleen ongelma pahenee, kun polynomeissa esiintyvien muuttujien määrä kasvaa. Jonkinlaisen käsityksen asiasta saa, jos tarkastellaan m:n muuttujan polynomien termien lukumäärää. Olkoot $x = (x_1, \ldots, x_m)$, $\alpha = (\alpha_1, \ldots, \alpha_m)$, ja merkitään

$$x^\alpha = x_1^{\alpha_1} \cdots x_m^{\alpha_m}.$$

Edelleen sanotaan, että monomin x^α aste on $\mathsf{deg}(x^\alpha) = \alpha_1 + \cdots + \alpha_m$ ja polynomin aste on luonnollisesti suurin sen monomien asteista. Jos sitten tarkastellaan yleistä q-asteista m:n muuttujan polynomia p, niin tämän esittämiseen tarvitaan

$$\frac{(m+q)!}{m!q!}$$

monomia. Esimerkiksi 10-asteisen 7 muuttujan polynomissa on lähes 20000 monomia. Kahden tällaisen polynomin kertolasku vaatii jo huomattavan monta operaatiota.[8]

Edellä on vaativuusanalyysissä tarkasteltu *pahimman tapauksen* analyysiä (worst case). On siis pyritty löytämään vaativuuden yläraja, joka pätee kaikille syötteille. Usein kuitenkin algoritmeilla on se hyvä tai kiusallinen ominaisuus, että "tyypillisesti" algoritmin vaatima työmäärä on huomattavasti pienempi kuin pahimman tapauksen työmäärä. Tällöin puhutaan *keskimääräisen tapauksen* (average case) analyysistä. Valitettavasti usein on kuitenkin hankalaa määritellä, mitä tarkoitetaan tyypillisellä syötteellä. Kuuluisa esimerkki tästä on lineaarisen optimoinnin *simplex-algoritmi*.[9] On osoitettu, että pahimmassa tapauksessa vaativuus ei ole polynomiaalinen. Kuitenkin käytännössä algoritmi on osoittautunut erittäin nopeaksi ja käyttökelpoiseksi.

Turingin kone

Teoreettinen malli kaikelle laskennalle on *Turingin kone*. Turing oli 40-luvulla von Neumannin ohella yksi tietokoneen teoreettisten perusteiden tutkimuksen pioneereista, ja näissä pohdinnoissaan hän päätyi Turingin

[8] Kuinka monta operaatiota tarvitaan? Kuinka monta monomia on tulopolynomissa?

[9] http://en.wikipedia.org/wiki/Simplex_algorithm

koneen käsitteeseen [5]. Voisi sanoa, että Turingin kone on eräänlainen tarkka määritelmä algoritmille. On myös esitetty muita ideoita siitä, miten laskenta ja algoritmi voitaisiin määritellä, mutta kaikki nämä ovat osoittautuneet ekvivalenteiksi Turingin koneen kanssa. On myös pohdittu, mikä merkitys Turingin koneen käsitteellä on ihmisen ajatteluprosessien kannalta. Voidaanko aivoja pitää Turingin koneena? Tätä, ja paljon muuta, on mielenkiintoisesti analysoitu kirjassa [6].

Numeerinen laskenta

Numeerinen laskenta poikkeaa monella tavalla yllä kuvatusta ”puhtaasta” tai symbolisesta laskennasta. Palataan tähän kirjoituksen seuraavassa osassa.

Viitteet

[1] J. Brown, *Kvanttitietokone*, Terra Cognita, 2001.

[2] J. Conway and R. Guy, *The book of numbers*, Copernicus, New York, 1996.

[3] D. Cox, J. Little, and D. O'Shea, *Ideals, varieties, and algorithms*, 3rd ed., Undergraduate Texts in Mathematics, Springer, New York, 2007, An introduction to computational algebraic geometry and commutative algebra.

[4] E. Heikkola ja P. Tarvainen, *Kokemuksia matematiikan hyödyntämisestä teollisuudessa*, Solmu (2010), no. 1, 14–17.

[5] A. Hodges, *Alan Turing, arvoitus*, Terra Cognita, 2000.

[6] D. Hofstadter, *Gödel, Escher, Bach: an eternal golden braid*, Basic Books Inc. Publishers, New York, 1979.

[7] T. W. Körner, *The pleasures of counting*, Cambridge University Press, 1996.

[8] Henri Lebesgue, *Leçons sur les Constructions Géométriques*, Les Grands Classiques Gauthier-Villars, Éditions Jacques Gabay, Sceaux, 2003.