Turmeleeko ohjelmointi nuorisomme?

Antti Laaksonen
Tietojenkäsittelytieteen laitos, Helsingin yliopisto
ahslaaks@cs.helsinki.fi

Uuden peruskoulun opetussuunnitelman mukaan syksystä 2016 alkaen matematiikan tunneilla opetetaan myös ohjelmointia. Tulisiko matematiikan ystävän olla huolestunut tästä muutoksesta? Miksi ohjelmointia opetetaan nimenomaan matematiikan tunneilla? Entä heikkeneekö koululaisten matematiikan osaaminen entisestään, kun tietokoneen käyttö lisääntyy?

Tämän kirjoituksen tarkoituksena on näyttää, mitä tekemistä ohjelmoinnilla on matematiikan kanssa. Syy siihen, miksi ohjelmointia opetetaan matematiikan tunneilla, on loppujen lopuksi yksinkertainen: ohjelmointi on matematiikan osa-alue. Tämän vuoksi ohjelmoinnilla on paikkansa matematiikan opetuksessa siinä missä vaikkapa algebralla ja analyysilla.

Eräs käsitys ohjelmoinnista on, että se on lähinnä pelailua ja viihteellistä puuhailua. Tällöin ohjelmointi toimisi uhkana ”kunnolliselle” matematiikalle. Tämä käsitys on kuitenkin yhtä oikea, kuin että geometria on joutavaa piirtelyä tai todennäköisyyslaskenta on noppien heittelyä ja uhkapeliä. Ohjelmointi on tärkeä matemaatikon työkalu, minkä lisäksi se opettaa syvällistä matemaattista ajattelua ja antaa uuden näkökulman perinteiseen matemaattiseen todistamiseen.

Ohjelmointi ei ole siis matematiikan vihollinen, vaan ohjelmoinnin osaaminen tukee muuta matematiikkaa ja päinvastoin. Tutustutaan seuraavaksi ohjelmointiin muutamien tehtävien kautta.

Simulointi

Tietokoneen vahvuutena on, että se pystyy laskemaan tehokkaasti ja luotettavasti. Siksi ohjelmoinnin avulla pystyy tutkimaan matemaattista ilmiötä simuloimalla sitä. Tarkastellaan esimerkiksi seuraavaa todennäköisyyslaskennan tehtävää:

Tehtävä 1: Lukujonossa $x_1, x_2, \ldots, x_n$ jokainen luku on satunnainen reaaliluku tasaisesta jakaumasta väliltä $[-10, 10]$. Mikä on odotusarvo lukujonon suurimmalle peräkkäisten lukujen summalle?

Esimerkiksi lukujonon $-1, 5, -2, 7, 1, -5, 3$ suurin summa on 11, joka saadaan valitsemalla luvut $5, -2, 7, 1$. Myös tyhjä summa on sallittu, minkä vuoksi suurin summa ei ole koskaan negatiivinen.

Merkitään $e(n)$ suurimman summan odotusarvoa, kun lukujonossa on n lukua. Odotusarvon $e(1)$ voi laskea kahdessa osassa. Jos $x_1 > 0$, suurin summa on x_1 ja odotusarvo on 5, ja jos $x_1 \leq 0$, suurin summa on tyhjä summa ja odotusarvo on 0. Molemmat tapaukset ovat yhtä todennäköisiä, joten $e(1) = 5/2$.

Odotusarvon $e(2)$ voi myös laskea samalla idealla. Jos $x_1 > 0$ ja $x_2 > 0$, suurin summa on $x_1 + x_2$ ja odotusarvo on 10. Jos $x_1 > 0$ ja $x_2 \leq 0$, suurin summa on x_1 ja odotusarvo on 5. Samoin tapahtuu symmetrisesti, jos $x_1 \leq 0$ ja $x_2 > 0$. Lopulta jos $x_1 \leq 0$ ja

$x_2 \leq 0$, odotusarvo on 0. Kaikki tapaukset ovat yhtä todennäköisiä, joten $e(2) = 20/4 = 5$.

Suuremmilla n:n arvoilla odotusarvon laskeminen perinteisen todennäköisyyslaskennan keinoin muuttuu hankalaksi, koska osatapausten määrä ja mutkikkuus kasvavat räjähdysmäisesti. Kuitenkin ohjelmoinnin avulla pystyy saamaan nopeasti käsityksen, millaisia suurempien tapausten odotusarvot ovat.

Seuraavan Python-kielisen koodin avulla pystyy tutkimaan suurimman summan odotusarvoa millä tahansa n:n arvolla. Funktio `laske` luo satunnaisen n luvun jonon ja laskee sen suurimman summan. Pääohjelma kutsuu funktiota annetun määrän kertoja ja tulostaa odotusarvon $e(n)$ arviona tulosten keskiarvon.

```
from random import uniform

def laske(n):
    p = t = 0
    for i in range(n):
        t = max(t,0)
        t += uniform(-10,10)
        p = max(p,t)
    return p

print "Montako lukua?",
n = input()
print "Montako kertaa?",
r = input()
s = 0
for i in xrange(r):
    s += laske(n)
print "Odotusarvo:", s/r
```

Ohjelman suoritus voi näyttää seuraavalta:

```
Montako lukua? 2
Montako kertaa? 10
Odotusarvo: 5.56135787434

Montako lukua? 2
Montako kertaa? 1000
Odotusarvo: 4.93420070081

Montako lukua? 2
Montako kertaa? 1000000
Odotusarvo: 5.00334192257
```

Ohjelman antaman tuloksen tarkkuus riippuu siitä, montako kertaa simulaatio toistetaan. Käytännössä esimerkiksi miljoona toistokertaa antaa hyvän arvion odotusarvon suuruudesta. Tällaiseen simulaatioon menee noin sekunti aikaa nykyaikaisella tietokoneella.

Seuraava taulukko sisältää odotusarvon $e(n)$ arvioita, kun $n = 1, 2, \ldots, 10$ ja simulaatio on toistettu miljoona kertaa kullekin n:n arvolle.

jonon pituus n	odotusarvon $e(n)$ arvio
1	2,49599
2	5,00334
3	7,19607
4	9,11295
5	10,78403
6	12,30602
7	13,71431
8	15,00966
9	16,22160
10	17,38309

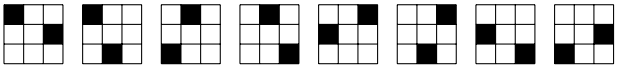
Tietokoneen simulaatio on hyvä apuneuvo ongelman tutkimisessa. Ensinnäkin simulaatio vahvistaa käsin lasketut arvot $e(1) = 5/2$ ja $e(2) = 5$, ja jos koettaisimme laskea lisää arvoja tarkasti, voisimme jälleen verrata tuloksia tietokoneen arvoihin. Todennäköisyyslaskenta on ihmiselle epäintuitiivista, mutta tietokone pystyy suorittamaan laajan toistokokeen ilmiöstä ja antaa siksi varmasti luotettavan tuloksen.

Johtaminen

Seuraavassa tehtävässä ohjelmointi tarjoaa oikotien kombinatoriikan kaavan johtamiseen. Ideana on selvittää ohjelmoinnin avulla pienten tapausten tuloksia ja päätellä niistä yleinen kaava.

Tehtävä 2: Monellako tavalla kaksi kuningatarta voidaan sijoittaa $n \times n$ -kokoiselle shakkilaudalle siten, että ne eivät uhkaa toisiaan?

Esimerkiksi jos $n = 3$, sijoitustapoja on 8:



Luonteva ensimmäinen askel tehtävän ratkaisussa on tutkia ratkaisujen määrää eri n:n arvoilla. Käsin tehtynä tämä olisi työlästä ja virhealtista, joten laskenta on järkevää antaa tietokoneen tehtäväksi. Seuraava raakaan voimaan perustuva koodi laskee kuningattarien sijoitustapojen määrän annetulle n:n arvolle.

```
print "Laudan koko:",
n = input()
t = 0
for x1 in range(n):
    for y1 in range(n):
        for x2 in range(n):
            for y2 in range(n):
                if x1 == x2 or y1 == y2:
                    continue
                if abs(x1-x2) == abs(y1-y2):
                    continue
                t += 1
print "Tulos:", t/2
```

Koodi käy läpi kaikki tavat valita kuningattarien paikat niin, että ensimmäisen kuningattaren paikka on (x_1, y_1) ja toisen kuningattaren paikka on (x_2, y_2). Koodi tarkistaa kuningattarien uhkaamisen kolmen ehdon perusteella. Ensinnäkin jos $x_1 = x_2$ tai $y_1 = y_2$, kuningattaret ovat samalla pysty- tai vaakarivillä. Lisäksi jos $|x_1 - x_2| = |y_1 - y_2|$, kuningattaret uhkaavat toisiaan sivusuuntaisesti. Jos kuningattaret eivät uhkaa toisiaan, koodi lisää laskurin t arvoa. Lopuksi koodi tulostaa arvon $t/2$, koska jokainen sijoitustapa tulee lasketuksi kahteen kertaan.

Koodin suoritus näyttää esimerkiksi seuraavalta:

```
Laudan koko: 3
Tulos: 8

Laudan koko: 8
Tulos: 1288

Laudan koko: 20
Tulos: 67260

Laudan koko: 50
Tulos: 2920400
```

Koodin perusteella syntyy seuraava taulukko tuloksista, kun laudan koko on $1, 2, \ldots, 10$:

laudan koko (n)	sijoitustavat
1	0
2	0
3	8
4	44
5	140
6	340
7	700
8	1288
9	2184
10	3480

Mikä sitten olisi yleinen kaava sijoitustapojen määrälle? Koodissa on neljä sisäkkäistä silmukkaa, minkä vuoksi hyvä oletus on, että kaava olisi 4. asteen polynomi muotoa $p(n) = an^4 + bn^3 + cn^2 + dn + e$.

Koodin ansiosta meillä on jo tuloksia pienille n:n arvoille, ja voimme päätellä kaavan niiden avulla. Kaavassa on viisi tuntematonta muuttujaa, joten tarvitaan viisi laskettua tulosta niiden päättelemiseen. Yksinkertaisin valinta on käyttää tuloksia $p(1), p(2), \ldots, p(5)$, joista saadaan yhtälöryhmä

$$\begin{cases} p(1) = 0 \\ p(2) = 0 \\ p(3) = 8 \\ p(4) = 44 \\ p(5) = 140 \end{cases}$$

eli auki kirjoitettuna

$$\begin{cases} a + b + c + d + e = 0 \\ 16a + 8b + 4c + 2d + e = 0 \\ 81a + 27b + 9c + 3d + e = 8 \\ 256a + 64b + 16c + 4d + e = 44 \\ 625a + 125b + 25c + 5d + e = 140 \end{cases}$$

Tämän yhtälöryhmän ratkaisu on

$$\begin{cases} a = 1/2 \\ b = -5/3 \\ c = 3/2 \\ d = -1/3 \\ e = 0 \end{cases}$$

joten polynomiksi tulee

$$p(n) = n^4/2 - 5n^3/3 + 3n^2/2 - n/3.$$

Tämä polynomi täsmää myös suurempiin laskettuihin arvoihin, esimerkiksi $p(8) = 1288$, kuten kuuluukin.

Nyt kun kombinatorinen kaava on löytynyt, sen voi vielä todistaa täsmällisesti. Todistaminen onkin mukavaa, kun kaava on valmiiksi tiedossa. Yksi mahdollisuus todistaa polynomikaava on näyttää, että se on sama kuin seuraava rekursiivinen funktio:

$$r(n) = \begin{cases} 0 & n = 1 \\ r(n-1) + 2(n-1)^2(n-2) & n > 1 \end{cases}$$

Rekursiivinen funktio laskee sijoitustapojen määrää kerros kerrallaan. Ensinnäkin jos $n = 1$, ei ole mitään tapaa sijoittaa kahta kuningatarta shakkilaudalle. Tapauksessa $n > 1$ mahdolliset tilanteet ovat

- molemmat kuningattaret ovat vasemman yläkulman $(n-1) \times (n-1)$ -shakkilaudan alueella, jolloin tapoja on $r(n-1)$,
- toinen kuningatar on alareunassa ja toinen kuningatar on oikeassa reunassa, jolloin tapoja on $(n-1)(n-2)$,
- toinen kuningatar on oikeassa alakulmassa ja toinen kuningatar on $(n-1) \times (n-1)$ -alueella, jolloin tapoja on $(n-1)(n-2)$, ja
- toinen kuningatar on alareunassa tai oikeassa reunassa (ei oikeassa alakulmassa) ja toinen kuningatar on $(n-1) \times (n-1)$ -alueella, jolloin tapoja on $2(n-1)(n-2)^2$.

Laskemalla yhteen nämä kaikki tapaukset saadaan tulokseksi $r(n-1) + 2(n-1)^2(n-2)$.

Tässä vaiheessa on hyvä varmistaa ohjelmoimalla, että rekursiivinen kaava on varmasti oikein:

```
def r(n):
    if n == 1:
        return 0
    else:
        return r(n-1)+2*(n-1)**2*(n-2)
```

```
print "Laudan koko:",
n = input()
print "Tulos:", r(n)
```

Koodi antaa samoja tuloksia kuin alkuperäinen koodi, eli kaikki on hyvin. Lisäksi koodin etuna on, että sillä voi laskea tehokkaasti myös suurempia tapauksia kuin alkuperäisellä koodilla. Esimerkiksi:

```
Laudan koko: 500
Tulos: 31042041500
```

Viimeinen vaihe todistuksessa on näyttää, että polynomikaava $p(n)$ ja rekursiivinen kaava $r(n)$ ovat samat. Tämä on suoraviivainen induktiotodistus, jonka lukija voi halutessaan tarkistaa.

Lopuksi polynomikaavan voi muuntaa ohjelmaksi:

```
def p(n):
    return (3*n**4-10*n**3+9*n**2-2*n)/6

print "Laudan koko:",
n = input()
print "Tulos:", p(n)
```

Tämän koodin avulla voimme laskea tehokkaasti hyvin suuria tapauksia:

```
Laudan koko: 123456789
Tulos: 1161528582630023586221568817641 24
```

Ohjelmointi ja matemaattinen päättely kulkivat käsi kädessä tässä tehtävässä. Ohjelmoinnin avulla pystyimme johtamaan polynomikaavan tehtävän ratkaisuun sekä tarkistamaan, että rekursiivinen kaava on oikein. Toisaalta matemaattisen päättelyn avulla pystyimme todistamaan kaavan sekä saimme aikaan ohjelmia, jotka toimivat huomattavasti nopeammin kuin alkuperäinen raakaan voimaan perustuva toteutus.

Yleensäkin ohjelmoinnissa matemaattinen päättely on avain tehokkaiden algoritmien luomiseen. Mitä paremmin ohjelmoija ymmärtää ongelman matemaattisen luonteen, sitä nopeamman koodin hän voi saada aikaan.

Todistaminen

Äskeisissä tehtävissä tietokone on toiminut matemaatikon apuna nopeasti laskevana työjuhtana. Mutta mitä tekemistä ohjelmoinnilla on syvällisemmän matemaattisen ajattelun kanssa? Viimeinen tehtävämme antaa näytteen tästä asiasta.

Tehtävä 3: Sinulle on annettu luvut $1, 2, 3, \ldots, n$ ja tehtäväsi on jakaa luvut kahteen joukkoon niin, että molemmissa joukoissa lukujen summa on sama. Onko tehtäväsi mahdollinen?

Esimerkiksi jos $n = 8$, tehtävä on mahdollinen, koska voidaan valita joukot $A = \{3, 7, 8\}$ ja $B = \{1, 2, 4, 5, 6\}$. Nyt molemmissa joukoissa lukujen summa on 18. Jos taas $n = 9$, tehtävä ei ole mahdollinen, koska lukujen $1, 2, 3, \ldots, 9$ summa on 45 eikä lukuja voi jakaa kahteen joukkoon, joiden summa olisi sama.

Ohjelmoijan näkökulmasta tehtävänä on laatia ohjelma, jossa käyttäjä antaa luvun n ja tietokoneen tulee esittää lukujen jako tai todeta, ettei se ole mahdollista. Seuraava koodi toteuttaa kyseisenlaisen ohjelman:

```
print "Anna luku:",
n = input()
s = sum(range(1,n+1))
if s%2 == 1:
    print "Mahdotonta!"
    exit()
A, B = [], []
u = s/2
for x in range(n, 0, -1):
    if x <= u:
        u -= x
        A.insert(0, x)
    else:
        B.insert(0, x)
print "Joukko A:", A
print "Joukko B:", B
```

Ohjelman suoritus voisi edetä seuraavasti:

```
Anna luku: 8
Joukko A: [3, 7, 8]
Joukko B: [1, 2, 4, 5, 6]
```

Tai seuraavasti:

```
Anna luku: 9
Mahdotonta!
```

Tai seuraavasti:

```
Anna luku: 12
Joukko A: [6, 10, 11, 12]
Joukko B: [1, 2, 3, 4, 5, 7, 8, 9]
```

Ohjelma laskee ensin summan $1+2+3+\cdots+n$ muuttujaan s. Jos summa on pariton, jako ei ole mahdollinen ja ohjelma päättyy. Muuten ohjelma jakaa luvut $1, 2, 3, \ldots, n$ joukkoihin A ja B siten, että kummankin joukon summaksi tulee $s/2$.

Muuttuja u pitää kirjaa, paljonko joukon A summasta puuttuu vielä. Ohjelma käy läpi muuttujalla x luvut $n, n-1, n-2, \ldots, 1$ ja lisää luvun x joukkoon A, jos se mahtuu sinne, ja muuten joukkoon B. Lopulta kaikki luvut on lisätty jompaankumpaan joukkoon ja muuttujan u arvona on 0, joten jako on onnistunut. Muuttuja u saavuttaa aina arvon 0, koska joka askeleella käytettävissä on kaikki luvut $1, 2, 3, \ldots, x$ jäljellä olevan summan muodostamiseksi.

Tämä ohjelma on tarkemmin katsoen epätavallisessa muodossa oleva matemaattinen todistus, koska se sisältää yleisen algoritmin, miten luvut $1, 2, 3, \ldots, n$ saadaan jaettua kahteen joukkoon, joiden summa on sama. Ohjelman etuna perinteiseen todistukseen verrattuna on, että se näyttää esimerkin jakotavasta annetulla n:n arvolla. Ohjelma tarjoaa siis näkymän todistuksen sisälle, ja todistuksessa käytettyä konstruktiota voi testata helposti millä tahansa n:n arvolla.

Lopuksi

Ohjelmoinnin opetuksen aloittamiseen kouluissa liittyy haasteita, koska ohjelmointi tuli opetussuunnitelmaan melko yllättäen ja kyseessä on monelle vieras aihealue. Kuitenkin asia, josta *ei* tarvitse murehtia, on, että ohjelmointi tekisi hallaa matematiikalle. Ohjelmointiin kätkeytyy rikas matemaattinen maailma, josta tämän kirjoituksen sisältö on antanut vain pientä esimakua.

Aiheeseen liittyvää

- Ohjelmointiputkan opas Python-ohjelmointiin: `http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=python_01`
- MAOLin Datatähti-ohjelmointikilpailu peruskoulun ja lukion oppilaille: `http://www.maol.fi/kilpailut/4tieteenkisat/datataehti/`
- Bentley, J.: *Programming Pearls*, 2. painos, Addison-Wesley, 1999.
- Skiena, S. ja Revilla, M.: *Programming Challenges*, Springer, 2003

Solmun matematiikkadiplomit

Peruskoululaisille tarkoitetut Solmun matematiikkadiplomit I–IX tehtävineen ovat tulostettavissa osoitteessa

`matematiikkalehtisolmu.fi/diplomi.html`

Opettajalle lähetetään pyynnöstä vastaukset koulun sähköpostiin. Pyynnön voi lähettää osoitteella

marjatta.naatanen(at)helsinki.fi tai juha.ruokolainen(at)helsinki.fi

Ym. osoitteessa on diplomitehtäville oheislukemistoa, joka varmasti kiinnostaa muitakin kuin diplomien tekijöitä:

Lukujärjestelmistä
Desimaaliluvut, mitä ne oikeastaan ovat?
Murtolukujen laskutoimituksia
Negatiivisista luvuista
Hiukan osittelulaista
Lausekkeet, kaavat ja yhtälöt
Äärettömistä joukoista
Erkki Luoma-aho: Matematiikan peruskäsitteiden historia
Funktiosta
Gaussin jalanjäljissä
K. Väisälä: Algebra
Yläkoulun geometriaa
Geometrisen todistamisen harjoitus
K. Väisälä: Geometria
Lukuteorian diplomitehtävät