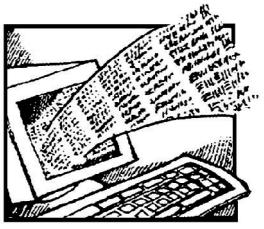


Kuinka löytää kaava lukujonolle?

Antti Laaksonen
Helsingin yliopisto
ahslaaks@cs.helsinki.fi

Olemme saaneet käsiimme lukujonon, joka alkaa $[1, -1, 3, 13, 29, \ldots]$, mutta emme tiedä sille *kaavaa* eli funktiota $f(x)$, jonka avulla lukujono voidaan esittää muodossa $[f(0), f(1), f(2), \ldots]$. Kuinka voisimme löytää tällaisen kaavan?

Tämä kirjoitus esittelee menetelmän, jonka avulla lukujonoa vastaava kaava löytyy aina olettaen, että se on *polynomi* eli muotoa $a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k$. Tämä oletus pätee melko monen vastaan tulevan lukujonon kohdalla. Menetelmää voi käyttää käsin, tai sen voi ohjelmoida tietokoneelle.

Menetelmän toiminta

Menetelmässä on ideana laskea lukujonon peräkkäisten jäsenten erotuksia, niiden erotuksia jne., kunnes syntyy lukujono, jonka jokainen jäsen on sama. Esimerkissämme meidän riittää laskea kaksi tasoa erotuksia, ennen kuin näin tapahtuu:

$$\begin{array}{rr} 1 & \\ & -2 \\ -1 & \\ & 4 \\ 3 & \\ & 10 \\ 13 & \\ & 16 \\ 29 & \end{array} \qquad \begin{array}{r} \\ 6 \\ \\ 6 \\ \\ 6 \\ \end{array}$$

Kun näin käy, on aihetta juhlaan: lukujono on ilmeisesti polynomi ja voimme myös jo päätellä sen asteen k sekä kertoimen a_k. Aste k on yhtä suuri kuin tasojen määrä, jotka muodostimme erotuksista. Tässä esimerkissä aste on 2. Kerroin a_k saadaan puolestaan kaavalla $x/k!$, missä x on viimeisellä tasolla toistuva luku. Tässä esimerkissä kerroin a_k on $6/2! = 3$. Tiedämme nyt siis, että polynomin korkeimman asteen termi on $3x^2$.

Muodostamme sitten uuden lukujonon vähentämällä saamamme termin jokaisesta lukujonon jäsenestä, eli jäsenestä $f(x)$ tulee $f(x) - a_k x^k$. Esimerkissämme uusi lukujono on $[1, -4, -9, -14, -19, \ldots]$. Tämän jälkeen laskemme erotukset uudelle lukujonolle:

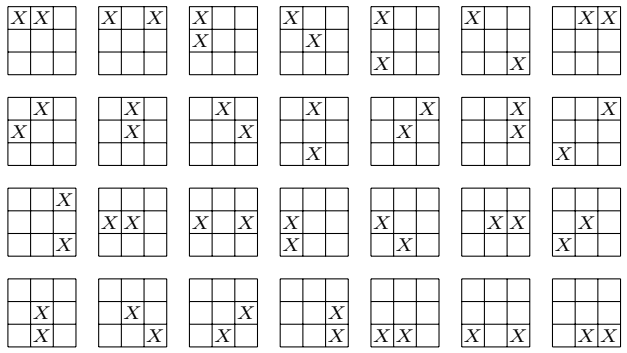
$$\begin{array}{rr} 1 & \\ & -5 \\ -4 & \\ & -5 \\ -9 & \\ & -5 \\ -14 & \\ & -5 \\ -19 & \end{array}$$

Tämä tarkoittaa, että aste k on 1 ja kerroin a_k on $-5/1 = -5$, eli polynomin toinen termi on $-5x$. Kun vähennämme tämän termin lukujonosta, tuloksena on $[1, 1, 1, 1, 1, \ldots]$. Tässä lukujonossa jokainen jäsen on valmiiksi sama, joten enää ei tarvitse laskea erotuksia, vaan voimme päätellä, että polynomin vakiotermi

on 1. Kokonaisuutena haettu polynomi on siis $f(x) = 3x^2 - 5x + 1$. Nyt kannattaa vielä testata, toimiiko polynomi varmasti. Esimerkiksi $f(1) = -1$ ja $f(4) = 29$, joten tilanne näyttää hyvältä.

Toinen esimerkki

Montako tapaa on asettaa $n \times n$ -shakkilaudalle kaksi ratsua niin, etteivät ne uhkaa toisiaan? Esimerkiksi tapauksessa $n = 3$ vastaus on 28, koska tavat ovat:



Laskemalla pieniä tapauksia ohjelmoinnin avulla selviää, että kyseinen lukujono alkaa $[0, 6, 28, 96, 252, 550, 1056, 1848, \dots]$.

Voimme nyt kokeilla, löytäisimmekö lukujonolle kaavan polynomina. Erotukset ovat seuraavat:

0				
	6			
6		16		
	22		30	
28		46		12
	68		42	
96		88		12
	156		54	
252		142		12
	298		66	
550		208		12
	506		78	
1056		286		
	792			
1848				

Lukujono näyttää olevan polynomi! Tasolla 4 erotukset ovat samat, joten polynomin aste on 4. Toistuva luku on 12, joten $a_4 = 12/4! = 1/2$. Tämän jälkeen voisimme selvittää muut polynomin termit vastaavasti.

Miksi menetelmä toimii?

Menetelmän idea on yksinkertainen, mutta miksi se toimii? Miten erotukset liittyvät asiaan, ja mistä ilmestyy kertoma jakajaksi?

Voimme ymmärtää menetelmän toiminnan tarkastelemalla, miten erotusten laskeminen vaikuttaa polynomin arvoihin. Oletetaan, että polynomin aste on k ja laskettavana on erotus $f(x+1) - f(x)$. Polynomin korkeimman asteen termin osalta erotus on

$$a_k(x+1)^k - a_k x^k = a_k(kx^{k-1} + A),$$

missä A kuvaa polynomin $(x+1)^k$ termejä, joiden aste on alle $k-1$. Tämän vuoksi erotuksessa termin x^{k-1} kertoimena on $a_k k$.

Voimme jatkaa vastaavaa päättelyä ja havaita, että seuraavalla tasolla kertoimena on $a_k k(k-1)$, sitä seuraavalla $a_k k(k-1)(k-2)$ jne., eli kertoma $k!$ syntyy taso kerrallaan termin $(x+1)$ kautta. Viimeisellä tasolla kerroin on $a_k k!$, joten saamme alkuperäisen kertoimen jakamalla tämän $k!$:lla. Lisäksi jokaisella tasolla erotukset poistavat polynomeista vakiotermit ja vähentävät muiden termien asteita yhdellä, joten viimeisellä tasolla jokaisessa kohdassa on luku $a_k k!$.

Entä jos polynomia ei ole?

Menetelmään liittyy yksi sudenkuoppa: se antaa *aina* jonkin polynomin, vaikka lukujonoa ei todellisuudessa voisi esittää polynomina. Esimerkiksi Fibonaccin lukujonoa ei voi esittää polynomina, mutta saamme silti lukujonolle $[0, 1, 1, 2, 3, 5, \dots]$ polynomin

$$\frac{x^5}{24} - \frac{13x^4}{24} + \frac{61x^3}{24} - \frac{119x^2}{24} + \frac{47x}{12}.$$

Tämä johtuu siitä, että n luvun lukujonon voi aina esittää polynomina, jonka aste on $n-1$. Tällöin erotusten laskemisessa viimeisellä tasolla on vain yksi erotus. Menetelmän antama tulos onkin luotettava vain silloin, kun viimeisellä tasolla on *useita* samoja erotuksia ja polynomin aste on pienempi kuin $n-1$.

Toisaalta polynomin muodostamisessa on aina riskinä, että polynomi perustuu vain lukujonon alkuosaan eikä meillä ole tietoa, mitä lukuja jonossa on sen jälkeen. Esimerkiksi teoriassa on mahdollista, että lukujonon sata ensimmäistä jäsentä noudattavat yksinkertaista polynomia, mutta niiden jälkeen tulee jotain muuta. Käytännössä esiintyvissä lukujonoissa tämä riski on kuitenkin äärimmäisen pieni.

Silti jos haluamme olla varmoja, että polynomi toimii aina, viimeinen vaihe on *todistaa* se oikeaksi. Tämä toki vaatii tietoa siitä, mistä lukujono on tullut, kuten mihin kombinatoriikan ongelmaan se liittyy.

Menetelmän ohjelmointi

Vaikka menetelmää voi käyttää mainiosti käsin, tämä on pidemmän päälle melko työlästä. Seuraavassa on menetelmän toteutus Python3-ohjelmana:

```
from fractions import Fraction

t = [1,-1,3,13,29]
a = 0

def etsi(t,c,f):
    if min(t) == max(t):
        return (t[0]/f,c)
    u = [t[i+1]-t[i]
         for i in range(len(t)-1)]
    return etsi(u,c+1,f*(c+1))

t = [Fraction(x) for x in t]

while True:
    p = etsi(t,0,1)
    if p == (0,0):
        break
    print(str(p[0])+"*x^"+str(p[1]))
    t = [t[i]-p[0]*(a+i)**p[1]
         for i in range(len(t))]
```

Ohjelman alussa listassa `t` annetaan lukujonon alkuosa. Muuttuja `a` puolestaan ilmaisee, mikä on lukujonon ensimmäisen jäsenen kohta. Esimerkissämme ohjelma tulostaa näin:

```
3*x^2
-5*x^1
1*x^0
```

Tämä vastaa polynomia $3x^2 - 5x + 1$.

Funktio `etsi` muodostaa erotuksia rekursiivisesti. Sen parametrit ovat lukujono `t`, polynomin asteluku `k` sekä vastaava kertoma `f`. Jos lukujonon pienin luku on yhtä suuri kuin suurin luku, kaikki luvut ovat yhtä suuria ja funktio päättyy. Muuten funktio muodostaa uuden lukujonon erotuksista ja kutsuu itseään rekursiivisesti.

Pääohjelmassa lukujonon `t` sisältö muutetaan ensin murtoluvuiksi (`Fraction`), jotta laskut suoritetaan tarkasti. Tämän jälkeen tulee silmukka, joka etsii joka kierroksella seuraavaksi suurimman polynomin termin ja tulostaa sen. Silmukka päättyy, kun sekä termin kerroin että aste ovat nolla, jolloin kaikki polynomin termit on löydetty.

Muuttamalla ohjelmaa voimme nyt muodostaa helposti myös kokonaisen kaavan ratsujen sijoitustapojen määrälle, jota laskimme toisessa esimerkissä. Annamme lukujonon ohjelmalle näin:

```
t = [0,6,28,96,252,550,1056,1848]
a = 1
```

Tässä tapauksessa `a` on 1, koska lukujonon ensimmäinen jäsen vastaa tapausta $n = 1$. Ohjelma tuottaa seuraavan tuloksen:

```
1/2*x^4
-9/2*x^2
12*x^1
-8*x^0
```

Ratsujen sijoitustapojen määrä $n \times n$ -shakkilaudalla on siis

$$\frac{n^4}{2} - \frac{9n^2}{2} + 12n - 8.$$

Tehtäviä

1. Todista, että äskeinen kaava ratsujen sijoitustapojen määrälle toimii aina.

2. Summa $1^k + 2^k + \cdots + n^k$ voidaan esittää aina polynomina, kun k on positiivinen kokonaisluku. Esimerkiksi kun $k = 2$, pätee

   $$1^2 + 2^2 + \cdots + n^2 = \frac{2n^3 + 3n^2 + n}{6}.$$

   Tutki menetelmän avulla, millaisia kaavoja saadaan suuremmille k:n arvoille.

3. Shakkiin liittyvät kombinatoriset ongelmat voi usein esittää polynomina. Millainen olisi esimerkiksi kaava, joka kertoo, monellako tavalla *kolme* ratsua voidaan sijoittaa shakkilaudalle niin, etteivät ne uhkaa toisiaan?