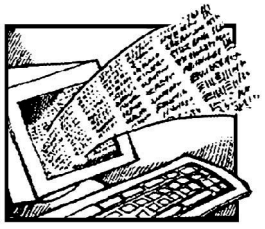

IMO vs. IOI – ongelmanratkojan kahdet olympialaiset

Antti Laaksonen
Helsingin yliopisto
ahslaaks@cs.helsinki.fi

Matematiikan olympialaiset IMO (*International Mathematical Olympiad*) on varmasti tuttu monelle Solmun lukijalle, mutta sen sisarkilpailu tietotekniikan olympialaiset IOI (*International Olympiad in Informatics*) on jäänyt vähemmälle huomiolle.

IMO ja IOI ovat yllättävänkin lähellä toisiaan, ja molemmissa kilpailuissa keskeinen teema on matemaattinen ongelmanratkaisu. Erona on, että IMO:ssa kirjoitetaan todistuksia kynällä ja paperilla, kun taas IOI:ssä ohjelmoidaan algoritmeja. Näihin olympialaisiin valmistautuminen on keino kehittää laaja-alaisesti ongelmanratkaisun taitoja.

Millaisia olympialaiset ovat?

IMO:ssa ja IOI:ssä perusidea on sama: jokainen maa lähettää olympialaisiin joukkueen, jossa on lukiolaisia (tai nuorempia) osallistujia. IMO:ssa joukkueessa on kuusi osallistujaa, kun taas IOI:ssä on neljä osallistujaa. Olympialaiset järjestetään joka vuosi eri maassa. Esimerkiksi vuonna 2020 IMO järjestetään Venäjällä ja IOI järjestetään Singaporessa.

Olympialaiset kestävät noin viikon, jonka kuluessa on kaksi kilpailupäivää. Vaikka mailla on joukkueet, kyseessä on yksilökilpailu ja jokainen osallistuja ratkoo tehtäviä itsenäisesti. Kilpailupäivien lisäksi olympialaisissa on paljon muutakin ohjelmaa, ja eri maista tulevilla osallistujilla on tilaisuus tutustua toisiinsa.

Nimien perusteella IMO:n aiheena on matematiikka, kun taas IOI:n aiheena on tietotekniikka. Tämä ei kerro kuitenkaan vielä paljon siitä, mitä kilpailuissa todella tehdään. IMO:n vakiintuneet aiheet ovat algebra, geometria, kombinatoriikka ja lukuteoria. IOI puolestaan keskittyy algoritmiikkaan, ja tehtävissä korostuu tietojenkäsittelytieteen matemaattinen puoli.

Kilpailujen taustaa

IMO:n historia ulottuu vuoteen 1959, jolloin järjestettiin ensimmäiset olympialaiset Romaniassa. Suomi osallistui ensimmäisen kerran vuonna 1965 ja on osallistunut säännöllisesti vuodesta 1973 alkaen. IMO on kasvanut vuosien kuluessa suureksi, ja vuonna 2019 mukana oli 621 osallistujaa 112 maasta.

IMO:n mallin mukaan alkoi syntyä muita tiedeolympialaisia, kuten fysiikan olympialaiset IPhO vuonna 1967 ja kemian olympialaiset IChO vuonna 1968. Tietotekniikka on nuorempi tulokas joukossa: ensimmäinen IOI järjestettiin vuonna 1989 Bulgariassa, ja Suomi on osallistunut kilpailuun vuodesta 1992 alkaen. Vuonna 2019 IOI:ssä oli mukana 327 osallistujaa 87 maasta.

Suomessa kilpailutoimintaa koordinoi MAOL ja pääasiallinen rahoittaja on Opetushallitus. Olympialaisten lisäksi toimintaan kuuluu paljon muutakin, kuten MAOLin neljän tieteen kisat, valmennusleirit ja pienemmät kansainväliset kilpailut. Valmentajat ovat tyy-

pillisesti yliopistojen työntekijöitä ja opiskelijoita, usein entisiä kilpailijoita. Tie olympialaisiin on pitkä, ja menestyminen vaatii määrätietoista harjoittelua.

Tehtävät ja arvostelu

IMO:ssa ja IOI:ssä on molemmissa kaksi kilpailupäivää ja kolme tehtävää ratkottavana kumpanakin päivänä. IMO:ssa aikaa päivän tehtäviin on 4,5 tuntia, kun taas IOI:ssä aikaa on 5 tuntia. Olympialaisten tehtävät valitaan paikan päällä kokouksessa, jossa on paikalla joukkueiden ohjaajia ja muita asiantuntijoita.

Oleellinen ero kilpailuissa on tapa, jolla tehtäviä ratkotaan. IMO:ssa tehtävän ratkaisu on todistus, joka kirjoitetaan kynällä ja paperilla. IOI:ssä puolestaan tehtävän ratkaisu on algoritmi, joka toteutetaan ohjelmointikielellä. IMO:ssa laskimet ja muut sähköiset apuneuvot ovat kiellettyjä, kun taas IOI:ssä jokaisella kilpailijalla on tietokone, jota saa käyttää vapaasti laskentaan.

Tehtävien ratkaisutapa vaikuttaa tehtävien asetteluun. IMO:ssa tehtävänä on tyypillisesti todistaa, että jokin pätee tai ei päde, tai etsiä ratkaisu, jonka pystyy ilmaisemaan lyhyesti kynällä ja paperilla. IOI:ssä taas keskeinen asia on algoritmien tehokkuus: ohjelmalle annetaan käsiteltäväksi suuri aineisto, ja sen täytyy selviytyä siitä nopeasti.

IMO:ssa arvostelu on useita päiviä kestävä prosessi, jossa eri maiden ohjaajat neuvottelevat pisteistä tuomariston kanssa. IOI:ssä puolestaan arvostelu on automaattinen ja perustuu ohjelmien testaamiseen. Molemmissa tavoissa on etunsa: IOI:ssä arvostelu on kaikille tasapuolinen eikä siinä ole tulkinnanvaraisuutta, mutta IMO:ssa pystyy antamaan paremmin pisteitä keskeneräisistä oikeansuuntaisista ratkaisuista.

Osallistuminen

Kumpiin olympialaisiin ongelmanratkaisusta kiinnostuneen kannattaisi tähdätä? Vastaus on yksinkertainen: *molempiin*. IMO ja IOI tukevat ja täydentävät toisiaan, ja moni menestyvä kilpailija myös osallistuu molempiin olympialaisiin. Yksi ääriesimerkki tästä on japanilainen kilpailija Yuta Takaya, joka voitti vuonna 2017 sekä IMO:n että IOI:n.

IMO ja IOI antavat yhdessä hyvän ongelmanratkojan peruskoulutuksen. IMO-tehtävät opettavat todistamista ja tärkeitä matematiikan tuloksia ja tekniikoita. IOI-tehtävät puolestaan opettavat ohjelmointia ja syvällisempää algoritmista ajattelua. Todistuksen kirjoittaminen ja algoritmin toteuttaminen antavat usein kaksi hieman erilaista näkökulmaa ongelmaan, mistä näemme seuraavaksi yhden esimerkin.

Tehtävä: Merkkijonot

Melko usein IMO:ssa ja IOI:ssä esiintyy tehtäviä, jotka voisi siirtää lähes sellaisenaan kilpailusta toiseen. Tuore esimerkki tällaisesta tehtävästä on IMO 2019:n tehtävä 5, jossa tarkastellaan seuraavaa prosessia:

Annettuna on merkkijono, jossa on n merkkiä ja jokainen merkki on H tai T. Joka vuorolla muutetaan vastakkaiseksi kohdassa $k > 0$ oleva merkki, missä k on merkkien H määrä. Näin jatketaan, kunnes jokainen merkki on T. Esimerkiksi jos annettu merkkijono on THT, muutoksista muodostuu ketju $THT \to HHT \to HTT \to TTT$, eli askelten määrä on 3.

IMO:ssa tehtävässä on kaksi kohtaa:

- Todista, että prosessi päättyy millä tahansa merkkijonolla.
- Määritä askelten määrän odotusarvo, kun merkkijonossa on n merkkiä ja kaikki 2^n merkkijonoa esiintyvät yhtä todennäköisesti.

Entä miten tehtävä muuttuisi, jos se olisikin IOI:ssä? Todennäköisesti näin:

- Tee tehokas algoritmi, joka laskee, montako askelta prosessissa on annetulla merkkijonolla.

Ratkaisemme seuraavaksi tehtävän tämän IOI-asettelun näkökulmasta.

Kohti ratkaisua

Ennen tehokkaan algoritmin laatimista meidän täytyy saada käsitys siitä, mistä prosessissa oikeastaan on kysymys. Yksi keino tähän on laatia ensin jokin hidas mutta toimiva raa'an voiman algoritmi. Voimme saada tietoa prosessista tämän algoritmin avulla.

Seuraavassa on C++-koodi, joka simuloi prosessia askel askeleelta. Koodi olettaa, että merkkijonon merkit ovat $s[1], s[2], \ldots, s[n]$, ja laskee muuttujaan c askelten yhteismäärän prosessissa.

```
int k = 0;
for (int i = 1; i <= n; i++) {
    if (s[i] == 'H') k++;
}
long c = 0;
while (k > 0) {
    if (s[k] == 'H') {
        s[k] = 'T'; k--;
    } else {
        s[k] = 'H'; k++;
    }
    c++;
}
```

Koodi etsii ensin muuttujaan k ensimmäisen muutettavan merkin kohdan. Tämän jälkeen koodi muuttaa merkkejä ja päivittää kohtaa k, kunnes lopuksi k on 0. Huomaa, että kohta k siirtyy aina viereiseen merkkiin vasemmalle tai oikealle, koska merkkien H määrä vähenee tai kasvaa yhdellä.

Koodin tehokkuudesta on vaikea sanoa vielä mitään, koska emme tiedä, miten kauan silmukan ehto $k > 0$ on voimassa. Voimme kuitenkin koodin avulla tutkia askelten määriä pienissä tapauksissa: esimerkiksi merkkijonolla $HHTTTHHT$ vastaus on 16. Kokeilemalla koodia useilla merkkijonoilla voimme saada käsitystä askelten määrän suuruusluokasta. Melko pian alkaa näyttää siltä, että vastaus on aina huomattavasti pienempi kuin kaikkien merkkijonojen määrä 2^n.

Saamme paljon lisää tietoa ongelmasta muuttamalla koodia hieman niin, että se tulostaa jokaisen vaiheen simulaatiossa. Nyt esimerkiksi merkkijonosta $HHTTTHHT$ syntyy seuraava tulostus:

```
HHTTTHHT
HHTHTHHT
HHTHHHHT
HHTHHTHT
HHTHTTHT
HHTTTTHT
HHHTTTHT
HHHHTTHT
HHHHHTHT
HHHHHHHT
HHHHHHTT
HHHHHTTT
HHHHTTTT
HHHTTTTT
HHTTTTTT
HTTTTTTT
TTTTTTTT
```

Tutkimalla tarkasti tällaisia tulostuksia eri merkkijonoille ongelman salat alkavat aueta. Näyttää siltä, että muuttuja k seilaa koodia suorittaessa merkkijonon puolelta toiselle ja joka kerta se kulkee aiempaa pidemmän matkan. Tämä on se hetki ongelmanratkaisussa, jolloin on aihetta juhlaan: olemme tehneet tärkeän havainnon, joka saattaa hyvinkin johtaa ratkaisuun.

Tehokas algoritmi

Jotta saamme aikaan toimivan tehokkaan algoritmin, meidän täytyy kuitenkin ymmärtää vielä tarkemmin, mistä prosessissa on kysymys. Miksi ja milloin muuttujan k suunta muuttuu ja miksi on varmaa, että prosessin päätteeksi jokainen merkki on T?

Tässä auttaa tarkastella erikseen kahta tapausta: Kun k liikkuu vasemmalta oikealle, merkkejä T korvataan merkillä H, kunnes vastaan tulee merkki H ja suunta vaihtuu. Kun taas k liikkuu oikealta vasemmalle, merkkejä H korvataan merkillä T, kunnes vastaan tulee merkki T ja suunta vaihtuu. Kun haluamme löytää kohdat, joissa suunta vaihtuu, meidän tulee siis vuorotellen paikantaa seuraava merkki H oikealta ja seuraava merkki T vasemmalta.

Jotta prosessi voi päättyä onnistuneesti, sen täytyy lähteä liikkeelle ”sopivasta” kohdasta, jossa merkkien T määrä vasemmalla on yhtä suuri kuin merkkien H määrä oikealla. Osoittautuu, että tämä kohta on sama kuin merkkien H yhteismäärä eli muuttujan k arvo alussa. Syynä on, että kun k ensimmäisen merkin joukossa on x kertaa merkki T, niin myös $n - k$ viimeisen merkin joukossa on x kertaa merkki H.

Nyt kaikki tarvittavat ideat alkavat olla kasassa ja voimme ryhtyä koodaamaan tehokasta algoritmia. Verrattuna ensimmäiseen algoritmiin meidän täytyy käytännössä onnistua tehostamaan osuuksia, joissa muuttuja k kulkee pitkän matkan samaan suuntaan. Tämä onnistuu ottamalla käyttöön kolme uutta muuttujaa: a ja b ovat nykyisen alueen vasen ja oikea reuna ja z kertoo liikkumissuunnan (0 = oikea, 1 = vasen). Monien vaiheiden ja virheiden korjaamisen jälkeen tuloksena on seuraava toimiva koodi:

```
int k = 0;
for (int i = 1; i <= n; i++) {
    if (s[i] == 'H') k++;
}
long c = 0;
int a = k, b = k;
int z = s[k] == 'H';
while (true) {
    if (z == 0) {
        a--;
        while (s[b] == 'T') b++;
    } else {
        b++;
        while (a >= 1 && s[a] == 'H') a--;
    }
    c += b-a-1;
    if (z == 1 && a == 0) break;
    z = 1-z;
}
```

Algoritmi määrittää k:n alkuarvon kuten ennenkin ja asettaa sen perusteella muut muuttujat. Tämän jälkeen alkaa pääsilmukka, jossa joka kierroksella alue laajenee joko oikealle tai vasemmalle ja liikuttu matka $b - a - 1$ lisätään askelten määrään. Silmukka päättyy, kun vasemmalle liikkumisen jälkeen $a = 0$.

Paitsi että algoritmi laskee tehokkaasti askelten määrän, se kertoo meille myös jotain muuta arvokasta. Koska algoritmi pysähtyy millä tahansa syötteellä ja il-

moittaa askelten määrän, saimme sivutuotteena ratkaistua IMO-tehtävän ensimmäisen kohdan: algoritmin olemassaolo *todistaa*, että askelten määrä on aina äärellinen.

Odotusarvon laskeminen

Entä IMO-tehtävän toinen kohta, jossa tulee määrittää askelten määrän odotusarvo? Kun meillä on koodi, joka laskee mille tahansa merkkijonolle askelten määrän, voimme alkajaisiksi laskea sen avulla odotusarvoja pienille n:n arvoille laittamalla koodin käymään läpi kaikki 2^n merkkijonoa. Saamme seuraavia tuloksia:

pituus n	odotusarvo
1	1/2
2	3/2
3	3
4	5
5	15/2
6	21/2
7	14
8	18

Nyt meillä on hyvää dataa, jonka avulla voimme arvuutella kaavaa odotusarvolle. Ehdokas kaavaksi löytyykin yllättävän helposti. Laskemalla taulukon lukujen erotuksia (ks. Solmu 1/2019) paljastuu, että odotusarvon kaava näyttää olevan polynomi

$$\frac{n^2+n}{4}.$$

Nyt olemme saaneet jo luultavasti selville vastauksen, mutta meillä ei ole vielä mitään tietoa, miksi vastaus on tuollainen. Tai ehkä on sittenkin: kaava näyttää hyvin samalta kuin summakaava

$$1+2+\cdots+n=\frac{n^2+n}{2},$$

jakaja vain on 4 eikä 2. Voisimme koettaa pyrkiä kohti todistusta, jossa olisi jotenkin lukujen summa.

Tässä vaiheessa on taas hyvä pysähtyä miettimään, miten muuttuja k oikeastaan liikkuu algoritmissamme. Saamme laskettua odotusarvon kaavalla

$$\frac{f(n)}{2^n},$$

missä $f(n)$ on summa kaikkien liikeratojen askelten määristä. Esimerkiksi kun $n=2$, liikeradat ovat:

- HH: 2 askelta vasemmalle
- HT: 1 askel vasemmalle
- TH: 1 askel oikealle, 2 askelta vasemmalle
- TT: 0 askelta

Niinpä $f(2)=2+1+3+0=6$, ja tapauksen $n=2$ odotusarvo on $6/2^2=3/2$.

Aiemmin huomasimme, että muuttujan k liikerata muodostuu osuuksista, jotka kulkevat vuorotellen eri suuntiin ja jokainen osuus on edellistä pidempi. Mutta voimme ajatella asiaa myös käänteisesti: jos on mikä tahansa liikerata, voimme saada muuttujan k kulkemaan sen, kunhan luomme sopivan merkkijonon, ja tietyllä n:n arvolla kyseinen merkkijono on yksikäsitteinen.

Tämän avulla pystymme johtamaan kaavan

$$f(n)=2f(n-1)+2^{n-1}n,$$

jossa on ideana, että n merkin merkkijonolla voidaan ensinnäkin muodostaa kaikki samat liikeradat kuin $n-1$ merkin merkkijonolla, mistä tulee $f(n-1)$ askelta. Lisäksi voidaan muodostaa liikeratoja, jossa viimeisen osuuden pituus on n ja sitä ennen on lyhempiä osuuksia. Näistä liikeradoista tulee $f(n-1)+2^{n-1}n$ askelta, koska $n-1$ merkin liikeratoja on 2^{n-1} erilaista ja jokaisesta tulee n askelta lisää viimeisen osuuden myötä. Laskemalla vielä vähän lisää selviää, että

$$\frac{f(n)}{2^n}=\frac{n^2+n}{4},$$

mikä olikin odotettavaa.

Aiheeseen liittyvää

- Peter Taylor: Comparisons of the IMO and IOI. *Olympiads in Informatics* 2012.
- IMO 2019 Problems. Saatavilla osoitteessa `https://www.imo-official.org/`.
- Antti Laaksonen: Kuinka löytää kaava lukujonolle? *Solmu* 1/2019.