582359 Algoritmit ongelmanratkaisussa (kevät 2013)
Viikon 2 ratkaisuja

1. Tehtävänä on laskea, monellako tavalla kaksi ratsua voidaan sijoittaa $n \times n$ -shakkilaudalle niin, että ne eivät uhkaa toisiaan. Esimerkiksi tapauksissa 3×3 ja 10×10 vastaukset ovat 28 ja 4662.

   (a) Ohjelmoi tehtävään brute-force-algoritmi. Mikä on sen aikavaativuus? Kuinka suuria tapauksia pystyt laskemaan algoritmilla?

   (b) Ohjelmoi tehtävään $O(1)$-aikainen algoritmi eli käytännössä mieti tilannetta matemaattisesti ja kehitä kaava, jolla vastauksen voi laskea suoraan. *Vihje*: laske ensin, moneenko suuntaan ratsu voi liikkua kussakin ruudukon ruudussa.

   Ratkaisu:

   (a) Tiedostossa `Ratsut.java` on brute-force-ratkaisu, jonka aikavaativuus on $O(n^4)$. Se toimii tehokkaasti, jos n on korkeintaan joitakin satoja.

   (b) Tiedostossa `Ratsut2.java` on $O(1)$-ratkaisu. Ratkaisun idea hahmottuu seuraavasta ruudukosta, jonka jokaisessa ruudussa lukee, monellako tavalla kyseisessä ruudussa olevaa ratsua voi uhata toinen ratsu. Seuraava ruudukko esittää 8×8 -shakkilautaa, mutta päättely yleistyy muihin tapauksiin.

2	3	4	4	4	4	3	2
3	4	6	6	6	6	6	3
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	6	3
2	3	4	4	4	4	3	2

   Ruudukosta havaitaan seuraavat asiat:

   - 4 kulmaruudussa tapoja on 2
   - 8 kulmaruudussa tapoja on 3
   - 4 kulmaruudussa tapoja on 4
   - joka reunalla $n - 4$ ruudussa tapoja on 4
   - joka reunalla $n - 4$ ruudussa tapoja on 6
   - kaikissa muissa $(n - 4) \times (n - 4)$ ruudussa tapoja on 8

   Ratsujen sijoitustapoja on kaikkiaan $n^2 \cdot (n^2 - 1)$, joten lopullisen tuloksen saa vähentämällä tästä uhkaavien sijoitustapojen määrän ja jakamalla kaiken 2:lla.

2. Muutetaan edellistä tehtävää niin, että ratsuja onkin kolme. Nyt mikään ratsu ei saa uhata toista asetelmassa. Ohjelmoi tehtävään brute-force-algoritmi, jolla voit laskea pienten tapausten vastauksia. Muodosta suoraan niiden perusteella polynomikaava esimerkiksi materiaalin menetelmällä.

   Voit halutessasi myös perustella matemaattisesti, miksi kyseinen polynomikaava antaa oikean vastauksen. Tämä on kuitenkin selvästi vaikeampaa kuin edellisessä tehtävässä.

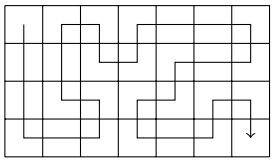
   Ratkaisu:

   Tiedostossa `Ratsut3.java` on brute-force-ratkaisu, ja tiedostossa `Ratsut4.java` on sen perusteella muodostettu $O(1)$-ratkaisu. Polynomikaava on:

$$\frac{n^6}{6} - \frac{9n^4}{2} + 12n^3 + \frac{85n^2}{3} - 164n + 180$$

   Kuitenkaan tämä kaava ei päde, jos $n < 4$. Tämän vuoksi nämä tapaukset käsitellään erikseen koodissa.

3. Tehtävänä on laskea, montako erilaista reittiä on ruudukon vasemmasta ylänurkasta oikeaan alanurkkaan. Reitin aikana täytyy käydä tasan kerran jokaisessa ruudussa, ja ruudukossa saa liikkua vasemmalle, oikealle, ylöspäin ja alaspäin. Tässä on yksi mahdollinen reitti 4×7 -ruudukossa:



Esimerkiksi tapauksessa 4×7 reittejä on yhteensä 111.

Ohjelmoi tehtävään brute-force-algoritmi. Kuinka suuria tapauksia pystyt laskemaan algoritmilla?

Ratkaisu:

Tiedostossa `Reitit.java` on brute-force-ratkaisu. Se laskee nopeasti tapauksen 4×7, mutta tapauksen 7×7 laskeminen vie useita minuutteja aikaa.

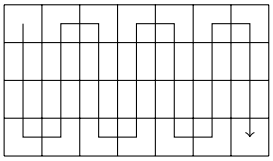
4. (a) Osoita, että jos sekä ruudukon korkeus että leveys on parillinen, niin edellisen tehtävän reittejä ei ole olemassa, ja muuten niitä on aina olemassa.

   (b) Tehosta algoritmiasi niin, että se pystyy sopivissa tilanteissa huomaamaan kesken reitin muodostuksen, ettei reittiä ole mahdollista muodostaa loppuun. Tällöin nykyisen hakuhaaran voi katkaista. Kuinka suuria tapauksia pystyt nyt laskemaan? *Tavoite*: tapauksen 7×7 tulisi ratketa hetkessä.

Ratkaisu:

(a) Osoitetaan aluksi, ettei reittiä voi muodostaa, jos sekä korkeus että leveys on parillinen. Ajatellaan ruudukkoa shakkiruudukkona, jossa vasen yläkulma on musta. Nyt reitti on muotoa M-V-M-V-$\cdots$, jossa M ja V tarkoittavat mustaa ja valkeaa ruutua. Reitissä on parillinen määrä ruutuja, joten viimeinen ruutu on V. Kuitenkin oikea alakulma on musta, joten viimeisen ruudun tulisi olla M. Tämä on ristiriita, joten reittiä ei ole olemassa.

Jos korkeus tai leveys on pariton, reitin voi muodostaa aina. Ideana on kulkea edestakaisin ruudukon rivejä sen mukaan, kumpi rivi on pariton. Jos molemmat rivit ovat parittomia, voi valita kumman tahansa. Esimerkin tapauksessa reitti on tämä:



(b) Tiedostossa `Reitit2.java` on tehostettu ratkaisu. Sen erona tehtävän 3 ratkaisuun on, että haku katkeaa aina, jos reitti päätyy ruudukon reunalle ja molemmilla puolilla reunalla on tyhjä ruutu. Tässä tilanteessa reitin jatkaminen loppuun on mahdotonta, koska sen eri puolilla olevista alueista ei pääse toisiinsa. Tämän optimoinnin ansiosta tapaus 7×7 ratkeaa muutamassa sekunnissa. Hakuun on mahdollista keksiä monenlaisia muitakin optimointeja.

5. *Heapin algoritmi* on erikoinen tapa käydä läpi joukon permutaatiot. Sen Java-toteutus on kurssisivulla tiedostossa `Heap.java`. Perustele, miksi Heapin algoritmi toimii.

Ratkaisu:

Tarkastellaan, miten algoritmi muuttaa taulukon n ensimmäistä lukua kutsussa `muodosta(n)`. Jos n on pariton, niin taulukon sisältö palautuu lopulta samaksi kuin alussa, eli esimerkiksi taulukosta [1, 2, 3, 4, 5] tulee [1, 2, 3, 4, 5]. Jos taas n on parillinen, niin jokainen luku siirtyy askeleen oikealle ja ensimmäinen luku siirtyy viimeiseksi, eli esimerkiksi taulukosta [1, 2, 3, 4, 5, 6] tulee [5, 1, 2, 3, 4, 6].

Tutkitaan tarkemmin tapausta, jossa taulukko on [1, 2, 3, 4, 5] eli n on pariton. Aluksi käsitellään taulukkoa [1, 2, 3, 4], jossa n on parillinen. Oletetaan, että taulukko muuttuu, kuten yllä on havaittu. Niinpä taulukon sisällöksi tulee [4, 1, 2, 3, 5]. Tämän jälkeen 4 ja 5 vaihtavat paikkaa ja tuloksena on [5, 1, 2, 3, 4]. Nyt

käsittelyyn tulee taulukko [5, 1, 2, 3] ja n on parillinen. Taulukko muuttuu, ja tuloksena on [3, 5, 1, 2, 4]. Nyt 3 ja 4 vaihtavat paikkaa ja tuloksena on [4, 5, 1, 2, 3]. Sitten taulukosta tulee [2, 4, 5, 1, 3] ja 2 ja 3 vaihtavat paikkaa. Sama jatkuu loppuun asti, jolloin taulukon sisältö on jälleen [1, 2, 3, 4, 5].

Tutkitaan sitten tapausta, jossa taulukko on [1, 2, 3, 4, 5, 6] eli n on parillinen. Aluksi käsitellään taulukkoa [1, 2, 3, 4, 5], jossa n on pariton. Oletetaan, että taulukko palautuu ennalleen yllä havaitun mukaisesti. Sitten 1 ja 6 vaihtavat paikkaa ja tuloksena on [6, 2, 3, 4, 5, 1]. Sitten 1 ja 2 vaihtavat paikkaa ja tuloksena on [6, 1, 3, 4, 5, 2]. Sitten 3 ja 2 vaihtavat paikkaa ja tuloksena on [6, 1, 2, 4, 5, 3]. Sama jatkuu loppuun asti, jolloin taulukon sisältö on [6, 1, 2, 3, 4, 5].

6. Lähtökohtana on lukujen $1 \ldots n$ permutaatio ja tehtävänä on järjestää luvut pienimmästä suurimpaan kääntöoperaatioiden avulla. Jokainen kääntöoperaatio kääntää valitun välin permutaatiosta takaperin.

   Esimerkiksi jos permutaatio on $(4, 2, 3, 1)$, lyhin mahdollinen kääntösarja on seuraava:

   $$(\underline{4, 2, 3, 1}) \rightarrow (1, \underline{3, 2}, 4) \rightarrow (1, 2, 3, 4)$$

   (a) Osoita, että minkä tahansa permutaation saa järjestykseen kääntöoperaatioilla.

   (b) Etsi lyhin kääntösarja seuraavalle permutaatiolle:

   $$(7, 13, 9, 4, 1, 14, 5, 3, 12, 10, 15, 2, 8, 6, 11)$$

   Vihje: IDA*-algoritmi on yksi hyvä lähestymistapa.

Ratkaisu:

Tiedostossa `Kaannot.java` on IDA*-algoritmin toteutus. Siinä on ideana suorittaa peräkkäisiä hakuja, joissa kääntösarjan pituus on rajoitettu. Algoritmissa on käytössä heuristiikka, joka arvioi, kuinka monta kääntöä vähintään tarvitaan valmiiseen ratkaisuun. Heuristiikan ansiosta haun voi usein katkaista varhaisessa vaiheessa, mikä nopeuttaa algoritmia merkittävästi.

Heuristiikassa on ideana laskea, monessako kohtaa taulukkoa on ”virhe” eli vierekkäin on kaksi lukua, joiden ei kuuluisi olla vierekkäin. Lisäksi taulukon alussa tai lopussa oleva väärä luku on myös virhe. Jos virheitä on v, niin alaraja kääntösarjan pituudelle on ylöspäin pyöristäen $v/2$, koska yhdellä käännöllä voi korjata korkeintaan kaksi virhettä. Esimerkiksi jos permutaatio on (3, 1, 2, 4, 5), niin $v = 3$, koska lukujen (3, 1) ja (2, 4) ei kuuluisi olla vierekkäin ja alussa oleva luku 3 on väärä. Niinpä missä tahansa ratkaisussa kääntöjä tarvitaan ainakin $3/2 = 1{,}5 \rightarrow 2$.