Satunnaisalgoritmit

Antti Tanhuanpää

25. maaliskuuta 2013

Johdanto

Satunnaisalgoritmit ovat algoritmeja, jotka hyödyntävät satunnaisuutta osana laskentaansa. Ensimmäisen tällaisen algoritmin kehitti Michael O. Rabin vuonna 1976 lähimmän parin ongelmaan (closest pair problem). Seuraavana vuonna Robert M. Solovay ja Volker Strassen kehittävät satunnaistetun alkulukutestin (primality test), johon ei vielä siihen aikaan tunnettu deterministista polynomista algoritmia. Deterministinen algoritmi kyseiseen ongelmaan löydettiin vasta 2002 eikä sekään vielä peittoa satunnaistettua.

Satunnaisalgoritmeista puhuttaessa on syytä tehdä ero satunnaistettujen ja probabilististen algoritmien välille. Satunnaistetut algoritmit hyödyntävät satunnaisuutta parantaakseen esimerkiksi pahimman tapauksen aikavaativuuttaan (satunnaistettu quick sort), mutta tuottavat aina oikean lopputuloksen. Probabilistiset algoritmit käyttävät laskennassaan satunnaisuutta, mutta eivät välttämättä tuota oikeaa lopputulosta (Monte Carlo -algoritmit) tai eivät tuota lopputulosta ollenkaan (Las Vegas -algoritmit).

Saattaa tuntua oudolta, miten tällaisista algoritmeista voisi olla mitään hyötyä, mutta käytännössä virheen salliminen on perusteltua, sillä laskenta tapahtuu aina fyysisillä laitteilla. Esimerkiksi alfahiukkaset ja kosminen säteily aiheuttavat virheitä laitteiden muistissa.

Tämä työ käsittelee ainoastaan probabilistisia algoritmeja ja niihin liittyvää laskennallista mallia, probabilistista Turingin konetta.

Probabilistiset algoritmit

Probabilistisia algoritmeja on kahta perustyyppiä: Las Vegas ja Monte Carlo -algoritmit. Las Vegas -algoritmit eivät välttämättä pysähdy, mutta pysähtyessään antavat aina oikean lopputuloksen. Monte Carlo -algoritmit puolestaan pysähtyvät aina, mutta niiden tuottama tulos on oikea vain jollain ennalta tiedetyllä todennäköisyydellä.

Motivoivana esimerkkinä esitetään alkion etsiminen listasta.

Alkion etsiminen listasta

Olkoon A eksponentiaalisen pitkä lista, jossa on 2^n alkiota. Puolet listan alkiosta on nollia ja puolet ykkösiä. Ongelmana on, kuinka löydämme listasta ykkösen (tai nollan)? Mikä tahansa deterministinen algoritmi toimii pahimassa tapauksessa ajassa $O(2^n/2) = O(2^{n-1})$, sillä saatamme joutua käymään läpi kaikki väärät alkiot ennen haluamamme alkion löytämistä.

Ratkaisun nopeuttamiseksi kehitämme probabilistisen algoritmin.

Las Vegas -algoritmi

Kehitetään Las Vegas -algoritmi alkion c löymiseksi listasta A. Se saa syötteenään listan A ja halutun alkion c.

```
las-vegas-search(A, c):
    while True:
        i = random(0, len(A) - 1)
        if A[i] == c:
            return i
```

Algoritmilla ei ole mitään yksittäistä pahinta syötettä, koska alkioita käydään läpi satunnaisessa järjestyksessä, ja aikavaativuus on listan pituudesta riippumaton. Hyvällä lykyllä löydämme haluamme alkion jo ensimmäisellä kierroksella, mutta voi käydä niin, että käymme loputtomasti läpi vääriä alkioita, ja pahimman tapauksen aikavaativuus onkin $O(\infty)$.

Las Vegas -algoritmi voidaan määritellä myös niin, että se pysähtyy aina, muttei välttämättä tuota lopputulosta. Annetaan algoritmille aiempien parametrien lisäksi jokin vakio k, joka kertoo, montako hakua listaan enimmällään tehdään.

```
las-vegas-search2(A, c, k):
    while k > 0:
        i = random(0, len(A) - 1)
        if A[i] == c:
            return i
        k = k - 1
```

Nyt algoritmimme pysähtyy aina ja algoritmin tuottama vastaus on aina oikea, jos se tuottaa vastauksen. Jos vastaus jää saamatta, laskentaan käytetty aika on heitetty hukkaan.

Monte Carlo -algoritmi

Monte Carlo -algoritmi samaiseen ongelmaan saa syötteenään samat parametrit, kuin edellinen Las Vegas -algoritmi.

```
monte-carlo-search(A, c, k):
    while k > 0:
        i = random(0, len(A) - 1)
        if A[i] == c:
            break
        k = k - 1
    return i
```

Algoritmi toimii syötteen pituudesta riippumattomassa ajassa, muttei välttämättä anna oikeaa vastausta. Sallimalla virhemarginaalin saamme stokastisia tuloksia. Virhemarginaali $\varepsilon = 2^{-k}$, koska jokaisella haulla todennäköisyys osua oikeaan on $\frac{1}{2}$. Todennäköisyys löytää annettu alkio $P[\text{löydetään } c] = 1 - \varepsilon$, joten varsin varsin vähäisellä hakujen määrällä päästään jo käytännössä luotettaviin tuloksiin. Todennäköisyys saada väärä tulos vielä 48 toiston jälkeen on pienempi, kuin saada Loton päävoitto kahdesti:

$$\begin{aligned} \varepsilon = 2^{-48} &\approx 3.6 \times 10^{-15} \\ P[\text{osua päävoittoon kahdesti}] = \binom{39}{7}^{-2} &\approx 4.3 \times 10^{-15} \end{aligned}$$

Seuraavaksi formuloimme mallin tutkia probabilistisia algoritmeja.

Probabilistinen Turingin kone

Probabilistinen Turingin kone (**PTM**) on epädeterministinen Turingin kone, jolla voidaan mallintaa satunnaisalgoritmien laskentaa. Toisin kuin tavallinen epädeterministinen Turingin kone (**NTM**), joka käy läpi kaikki laskentapuun haarat, PTM valitsee jokaisella epädeterministisellä askeleella satunnaisesti ("kolikkoa heittämällä") toisen kahdesta mahdollisesti konfiguraatiosta, mikä tekee siitä mahdollisen toteuttaa käytännössä.

PTM voidaan nähdä deterministisen Turingin koneen (**DTM**) varianttina, jolla on käytössään loputon nauha tasaisesti jakautuneita satunnaisbittejä, josta se lukee valitessaan seuraavaa konfiguraatiotaan.

Probabilistisen Turingin koneen M jokaiseen laskennan haaraan b sidotaan todennäköisyys $P[b] = 2^{-n}$, missä n on epädeterminististen askelten määrä haarassa

b, jolla M valitsee kyseisen haaran. Näin voimme määritellä todennäköisyyden sille, että M hyväksyy tai hylkää syötteen ω.

$$P[M \text{ hyväksyy syötteen } \omega] = \sum_{b \text{ hyväksyy}} P[b] \quad \text{ja}$$
$$P[M \text{ hylkää syötteen } \omega] = 1 - P[M \text{ hyväksyy syötteen } \omega].$$

Toisin sanoen todennäköisyys hyväksyä syöte on summa yli kaikkien hyväksyvien haarojen todennäköisyyksien ja todennäköisyys hylätä tämän komplementti.

Kuten deterministisetkin Turingin koneet, PTM tunnistaa kielen, jos se hyväksyy kieleen kuuluvat syötteet ja hylkää siihen kuulumattomat. Probabististen koneiden kohdalla kuitenkin sallitaan virhemarginaali $0 \leq \varepsilon < \frac{1}{2}$ ja sanotaan, että M tunnistaa kielen A virhemarginaalilla ε, jos

$$\begin{aligned} \omega \in A &\implies P[M \text{ hyväksyy syötteen } \omega] \geq 1 - \varepsilon \quad \text{ja} \\ \omega \notin A &\implies P[M \text{ hylkää syötteen } \omega] \quad \geq 1 - \varepsilon. \end{aligned}$$

Määriteltyämme laskinnallisen mallin satunnaisalgoritmeille on mielekästä puhua aikavaativuusluokista.

Probabilistiset vaativuusluokat

Rajallisen virheen probabilistinen polynomiaalinen aikavaativuus

Rajallisen virheen probabilistinen polynomiaalinen aikavaativuus (bounded-error probabilistic polynomial time eli **BPP**) on niiden kielten luokka, jotka PTM tunnistaa pahimman tapauksen polynomiaalisessa ajassa virhemarginaalilla $\varepsilon = \frac{1}{3}$.

Virhemarginaalin valinta on mielivaltainen, kunhan $0 \leq \varepsilon < \frac{1}{2}$, mikä tullaan osoittamaan vahvistuslemman (amplification lemma) käsittelyn yhteydessä.

BPP on laajimpia käytännössä hyödyllisiä vaativuusluokkia. Suurimmalle osalle tämän luokan ongelmista on olemassa probabilistinen algoritmi, joka voidaan toteuttaa tehokkaasti nykyisillä tietokoneilla.

Monte Carlo -algoritmit kuuluvat tähän luokkaan.

Satunnaistettu polynominen aikavaativuus

Satunnaistettu polymoninen aikavaativuusluokka (randomized polynomial time, **RP** ja **co-RP**) poikkeaa luokasta BPP sillä, että siinä sallitaan virhe ainoastaan yhteen suuntaan. Toisin sanoen, jos oikea ratkaisu on hyväksyä syöte, se hyväksytään virhemarginaallilla $0 \leq \varepsilon \leq \frac{1}{2}$, mutta jos oikea ratkaisu on hylätä syöte, se hylätään aina (ja toisinpäin luokassa co-RP).

Virheetön probabilistinen polynomiaalinen aikavaativuus

Virheetön probabilistinen polynomiaalinen aikavaativuus (zero-error probabilistic polynomial time, **ZPP**) on luokkien RP ja co-RP leikkaus.

Jos Las Vegas -algoritmit määritellään, kuten jälkimmäinen esimerkkimme, ne kuuluvat tähän luokkaan.

Probabilististen vaativuusluokkien suhteet

Vaativuusluokka ZPP on P:n laajennus, joten P on ZPP:n osajoukko. Toisaalta ZPP on luokkien RP ja co-RP leikkauksena selvästikin molempien osajoukko, jotka puolestaan ovat BPP:n erikoistapauksina sen osajoukkoja. Saadaan siis seuraavat yhteydet:

$$P \subseteq ZPP \subseteq \quad RP \quad \subseteq BPP$$
$$P \subseteq ZPP \subseteq co\text{-}RP \subseteq BPP$$

Mielenkiintoista onkin, onko $P \neq BPP$. Tätä ei ole toistaiseksi onnistuttu todistamaan, ja yleisesti uskotaankin, ettei näin ole. Luultavasti siis kaikki esitellyt luokat ovat keskenään samoja.

Eikö satunnaisuus tuokaan lisää laskentavoimaa? Ehkä pystymme simuloimaan satunnaisuutta riittävän hyvin deterministisillä pseudosatunnaisgeneraattoreilla, joiden tuottamia arvoja polynomiaikainen DTM ei pysty erottamaan aidosta satunnaisesta.

Luokkien BPP ja NP suhdetta ei tunneta.

Vahvistuslemma

Vahvistuslemman avulla minkä tahansa BPP-kielen virhemarginaali ε saadaan kutistettua mielivaltaisen pieneksi ilman, että aikavaativuus kasvaa.

Olkoon $0 \leq \varepsilon < \frac{1}{2}$. Kaikilla polynomisesti syötteen pituudesta n riippuvilla arvoilla *poly(n)* voidaan polynomiselle probabilistiselle Turingin koneelle M_1 virhemarginaalilla ε muodostaa ekvivalentti polynominen PTM M_2 virhemarginaalilla $0 \leq \varepsilon' = 2^{-poly(n)} \leq \varepsilon$.

Vahvistuslemman todistus sivuutetaan. Ideana on suorittaa algoritmia polynomiaalisen monta kertaa ja katsoa, hylätäänkö vai hyväksytäänkö syöte suurimmalla osalla kerroista. Virhemarginaali pienenee eksponentiaalisessa suhteessa suoritusten määrään nähden, mikä voidaan osoittaa Chernoffin rajaa käyttämällä. Tarvittavien toistojen määrä riippuu virhemarginaaleista ε ja ε', mutta pysyy polynomisessa suhteessa syötteen pituuteen nähden.

Yhteenveto

Vaikka satunnaistetut laskennalliset mallit eivät todennäköisesti tarjoakaan teoreettisia hyötyjä, käytännössä niillä voidaan saada huomattavia nopeutuksia.

Lähteet

- Sipser, Michael 2006. Introduction to the Theory of Computation. 2nd international edition. USA: Course Technology.
- Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford 2011. Introduction to Algorithms. 2nd edition. USA: MIT Press.
- Lynch, Nancy 2011. MIT 6.045: Automata, Computability, and Complexity (GITCS), Class 17. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/lecture-notes/MIT6_045JS11_lec17.pdf