Randomizovani algoritmi

Algoritmi koje smo do sada razmatrali bili su deterministički — svaki naredni korak bio je unapred određen. Kad se deterministički algoritam izvršava dva puta sa istim ulazom, način izvršavanja je oba puta isti, kao i dobijeni izlazi. Za razliku od njih *randomizovani algoritmi* (eng. randomized algorithms) sadrže korake koji zavise ne samo od ulaza, nego i od nekih slučajnih događaja. Naime, randomizovani algoritam koristi slučajne brojeve koji utiču na izbore koji se prave tokom njegovog izvršavanja. Stoga njegovo ponašanje, koje se uobičajeno meri kroz njegovo vreme izvršavanja ili kvalitet izlaza, za fiksiranu vrednost ulaza varira od jednog do drugog izvršavanja.

Koreni randomizovanih algoritama leže u Monte Karlo metodi koja se koristi u numeričkoj analizi. Danas randomizovani algoritmi nalaze sve veći broj primena u raznim oblastima. Naime, randomizacija u mnogim slučajevima omogućava konstruisanje jednostavnijih algoritama, kao i algoritama koji su efikasniji od najboljeg poznatog determinističkog algoritma.

Postoji veliki broj varijacija randomizovanih algoritama. Mi ćemo u ovom materijalu razmotriti dve vrste njih: *Las Vegas algoritme* koji uvek daju tačan rezultat, ali im vreme izvršavanja nije garantovano i *Monte Karlo algoritme* koji mogu dati netačan rezultat sa određenom (tipično jako malom) verovatnoćom, ali se uvek zaustavljaju u konačnom vremenu i vreme izvršavanja im je bolje nego za najbolji deterministički algoritam. Primetimo da se algoritmi tipa Las Vegas mogu izvršavati brzo, ali se mogu izvršavati i proizvoljno dugo. Oni su korisni samo ako im je *očekivano* vreme izvršavanja ograničeno.

Randomizovani algoritmi su posebno korisni u situacijama kada se "suočavamo" sa protivnikom koji pokušava da prosledi loš ulaz u algoritam, kao što je recimo slučaj sa primenama u kriptografiji.

Razmotrimo jedan jednostavan problem. Dat je neuređen niz a od $n \geq 2$ celobrojnih vrednosti u kome polovina elemenata ima vrednost 0, a polovina elemenata vrednost 1. Potrebno je pronaći element niza koji ima vrednost 0. Deterministički algoritam za rešavanje ovog problema bi prošao redom kroz elemente niza i nakon najviše $n/2 + 1$ koraka našao element niza čija je vrednost jednaka 0. Dakle, njegova vremenska složenost bila bi $O(n)$. Razmotrimo kako bi izgledao randomizovani algoritam za rešavanje ovog problema. Algoritam tipa Las Vegas bi *na slučajan način* birao element niza a i proveravao da li je njegova vrednost jednaka 0 i ove korake ponavljao sve dok ne bismo naišli na element sa vrednošću 0. Razmotrimo kakve su performanse ovog algoritma. On sa verovatnoćom 1 vraća ispravan rezultat, ali broj iteracija algoritama varira i može biti proizvoljno veliki. Analizirajmo koliko je očekivano vreme izvršavanja algoritma. Svaki nasumično odabrani element sa verovatnoćom $1/2$ ima vrednost 0, a sa verovatnoćom $1/2$ vrednost 1. Verovatnoća da je u i-toj iteraciji *po prvi*

put pronađen element sa vrednošću 1 je $(1/2)^i$. Stoga je očekivani broj iteracija algoritma jednak:

$$\lim_{n\to\infty} \sum_{i=0}^{n} \frac{i}{2^i} = 2$$

Naime, ako gornju sumu označimo sa S, važi:

$$S = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \dots$$

Odavde sledi:

$$\frac{S}{2} = \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \frac{4}{32} + \dots$$

Oduzimanjem ove dve jednakosti dobijamo:

$$\frac{S}{2} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots = \sum_{i=0}^{\infty} (\frac{1}{2})^i - 1 = \frac{1}{1-1/2} - 1 = 2 - 1 = 1$$

odakle dobijamo da je $S = 2$. Dakle, očekivani broj iteracija u kojem bi bio pronađen element sa vrednošću 0 je 2. Stoga je očekivana vremenska složenost ovog algoritma $O(1)$.

Razmotrimo kako bi izgledao Monte Karlo algoritam za rešavanje ovog problema. On bi na slučajan način birao element niza a i proveravao da li je vrednost elementa 0. Ukoliko to nije slučaj iznova bi ponavaljao ovaj postupak, ali maksimalno m puta. Primetimo da u ovom slučaju ne postoji garancija da će algoritam vratiti korektan rezultat. Naime, može se desiti da se u svih m iteracija petelje dobije element sa vrednošću 1. Primetimo i da je verovatnoća da se nakon k iteracija pronađe traženi element jednak $1 - (\frac{1}{2})^k$. Vremenska složenost ovog algoritma je ograničena maksimalnim dopuštenim brojem iteracija m, te iznosi $O(1)$, ali uspešnost algoritma nije garantovana.

Primetimo da se Las Vegas algoritam uvek jednostavno može pretvoriti u Monte Karlo algoritam tako što čemo ga pustiti da radi neko fiksirano vreme i vratiti proizvoljni odgovor ako se za to vreme ne zaustavi. Ovo funkcioniše zato što možemo ograničiti verovatnoću da algoritam radi duže od fiksiranog vremenskog limita. Ako je raspoloživ deterministički način za utvrđivanje korektnosti izlaza, onda je moguće transformisati Monte Karlo algoritam u Las Vegas algoritam.

Određivanje broja iz gornje polovine

Pretpostavimo da je dat skup brojeva $x_0, x_1, \ldots, x_{n-1}$ i da među njima treba izabrati neki broj iz gornje polovine, odnosno broj koji je veći ili jednak od bar $n/2$ ostalih. Na primer, potrebno je izabrati dobrog studenta, pri čemu je kriterijum prosečna ocena. Jedna mogućnost je uzeti najveći broj (koji je uvek u gornjoj polovini). Već smo videli da je za određivanje maksimuma potrebno $n-1$ upoređivanja. Druga mogućnost je započeti sa izvršavanjem algoritma za nalaženje maksimuma, i zaustaviti se kad se prođe polovina brojeva. Broj koji je veći ili jednak od jedne polovine brojeva je sigurno u gornjoj polovini. Algoritam zahteva oko $n/2$ upoređivanja. Može li se ovaj posao obaviti efikasnije? Nije teško pokazati da je nemoguće garantovati da broj pripada gornjoj polovini ako je izvršeno manje od $n/2$ upoređivanja. Prema tome, opisani algoritam je optimalan.

Ovaj algoritam je, međutim, optimalan samo ako insistiramo na *garanciji*. U mnogo slučajeva garancija nije neophodna, dovoljna je pristojna verovatnoća da je rešenje tačno. Na primer, kod heš tabela nije moguće garantovati da do kolizija neće doći, ali postoji način za rešavanje problema izazvanih pojavom kolizija (operacije sa heš tabelama se takođe mogu smatrati randomizovanim algoritmima). Ako odustanemo od garancije, onda postoji bolji algoritam za nalaženje elementa iz gornje polovine. Izaberimo na slučajan način dva broja x_i i x_j iz skupa, tako da je $i \neq j$. Pretpostavimo da je $x_i \geq x_j$. Verovatnoća da slučajno izabrani broj iz skupa pripada gornjoj polovini je bar $1/2$ (ona će biti veća od $1/2$ ako je više brojeva jednako medijani). Verovatnoća da ni jedan od brojeva x_i, x_j ne pripada gornjoj polovini je najviše $1/4$. Zbog $x_i \geq x_j$ ova verovatnoća jednaka je verovatnoći da x_i ne pripada gornjoj polovini. Prema tome, verovatnoća da x_i pripada gornjoj polovini je bar $3/4$.

Verovatnoća $3/4$ da je dobijeni rezultat tačan obično nije dovoljna. Međutim, opisani pristup može se uopštiti. Na slučajan način biramo k brojeva iz skupa i određujemo najveći od njih. Na isti način kao u specijalnom slučaju, zaključujemo da najveći od k elemenata pripada gornjoj polovini sa verovatnoćom najmanje $1 - 2^{-k}$ (on ne pripada gornjoj polovini akko gornjoj polovini ne pripada ni jedan od izabranih brojeva, što je događaj sa verovatnoćom najviše 2^{-k}). Na primer, ako je $k = 10$, verovatnoća uspeha je približno 0.999, a za $k = 20$ ta verovatnoća je oko 0.999999. Ako je pak $k = 100$, onda je verovatnoća greške za sve praktične svrhe zanemarljiva. Imamo dakle algoritam koji bira broj iz gornje polovine sa proizvoljno velikom verovatnoćom, a koji izvršava mali broj upoređivanja nezavisno od veličine ulaza. Ovo je primer Monte Karlo algoritma.

Slučajni brojevi

Bitan element randomizovanih algoritama je generisanje slučajnih brojeva. Tradicionalni način za generisanje slučajnih brojeva se često realizuje bacanjem kockice. Međutim, potrebno je imati efikasne računarske metode za rešavanje ovog problema. Deterministička procedura generiše brojeve na fiksirani način, pa tako

dobijeni brojevi ne mogu biti slučajni u pravom smislu te reči. Ti brojevi su međusobno povezani na sasvim određeni način. Na sreću, to u praksi nije veliki problem: dovoljno je koristiti tzv. *pseudoslučajne brojeve*. Ti brojevi generišu se determinističkom procedurom (pa dakle nisu pravi slučajni brojevi), ali procedura generisanja je dovoljno složena, da je teško uočiti međuzavisnosti između tih brojeva.

Jedan od načina za dobijanje pseudoslučajnih brojeva je *linearni kongruentni metod*. Prvi korak je izbor celog broja X_0 kao prvog člana niza, slučajnog broja izabranog na neki nezavisan način (na primer, tekuće vreme u mikrosekundama). Ostali brojevi izračunavaju se na osnovu diferencne jednačine

$$X_n = (aX_{n-1} + b) \mod m,$$

gde su a, b i m konstante, koje je potrebno pažljivo izabrati. Za vrednost parametra m se često bira vrednost $2^{31}-1$. I pored pažljivog izbora, ovakvi nizovi često nisu dovoljno „slučajni“. Alternativa je korišćenje diferencnih jednačina višeg reda (sa zavisnošću od većeg broja prethodnih članova).

Na ovaj način dobija se niz brojeva iz opsega od 0 do $m-1$. Ako su potrebni slučajni brojevi iz opsega od 0 do 1, onda se članovi ovog niza mogu podeliti sa m.

Bojenje elemenata skupa

Ideja randomizovanih algoritama tesno je povezana sa izvođenjem dokaza. Korišćenje verovatnoće za dokazivanje kombinatornih tvrđenja je moćna tehnika. U osnovi, ako se dokaže da neki objekat iz skupa objekata ima verovatnoću veću od nule, onda je to indirektan dokaz da postoji objekat sa tim osobinama. Ova ideja može se iskoristiti za konstrukciju randomizovanog algoritma. Pretpostavimo da tražimo objekat sa nekim osobinama, a znamo da ako generišemo slučajni objekat, on će zadovoljavati željeni uslov sa verovatnoćom većom od nule (što je probabilistički dokaz da traženi objekat postoji). Mi zatim pratimo probabilistički dokaz, generišući slučajne događaje kad je potrebno, i na kraju nalazimo željeni objekat se nekom pozitivnom verovatnoćom. Ovaj postupak može se ponavljati više puta, sve do uspešnog pronalaženja željenog objekta.

Razmotrimo jedan ovakav algoritam. Neka je S skup od n elemenata, i neka je $S_1, S_2, \ldots, S_k$ kolekcija njegovih različitih podskupova, od kojih svaki sadrži tačno r elemenata, $r \geq 2$, pri čemu je $k \leq 2^{r-2}$. Potrebno je obojiti svaki element skupa S jednom od dve boje, crvenom ili plavom, tako da svaki podskup S_i sadrži bar jedan plavi i bar jedan crveni element.

Primer ulaza za ovaj problem prikazan je u narednoj tabeli: skup S sadrži $n = 8$ elemenata, razmatramo njegovih $k = 3$ podskupa, a svaki od tih podskupova sadrži po $r = 4$ elementa. Primetimo da važi uslov zadatka da je $k = 3 \leq 2^2 = 2^{4-2} = 2^{r-2}$.

podskup/element	1	2	3	4	5	6	7	8
1		x	x		x	x		
2			x	x		x	x	
3	x			x		x	x	

Bojenje koje zadovoljava taj uslov zvaćemo *ispravnim bojenjem*. Ispostavlja se da pod navedenim uslovima ispravno bojenje uvek postoji. U prethodnom primeru bilo bi dovoljno obojiti element 6 u plavo, elemente 5 i 7 u crveno, a ostale elemente kako god želimo.

Jednostavan randomizovani algoritam dobija se prepravkom probabilističkog dokaza postojanja takvog bojenja:

> *Obojiti svaki element skupa S slučajno izabranom bojom, plavom ili crvenom, nezavisno od bojenja ostalih elemenata.*

Jasno je da ovaj algoritam ne daje uvek ispravno bojenje.

Izračunajmo verovatnoću neuspeha. Verovatnoća da su svi elementi skupi S_i obojeni crveno je 2^{-r}, a verovatnoća da su svi obojeni istom bojom (crvenom ili plavom) je $2 \cdot 2^{-r} = 2^{1-r}$. Označimo sa A slučajni događaj da je neki od skupova S_i neispravno obojen: on je unija svih slučajnih događaja A_i da je skup S_i neispravno obojen, za $i = 1, 2, \ldots, k$. Važi nejednakost:

$$P(A) \leq \sum_{i=1}^{k} P(A_i) = \sum_{i=1}^{k} 2^{1-r} = k2^{1-r} \leq 2^{r-2}2^{1-r} = \frac{1}{2}.$$

Time je dokazano da ispravno bojenje postoji (u protivnom bi verovatnoća neispravnog bojenja bila tačno 1). Pored toga, vidi se da je ovaj randomizovani algoritam dobar. Ispravnost zadatog bojenja lako se proverava: proveravaju se elementi svakog podskupa dok se ne pronađu dva različito obojena elementa. Verovatnoća uspešnog bojenja $p = 1 - P(A)$ je bar $1/2$. Ako se u jednom pokušaju ne dobije neispravno bojenje, postupak (bojenje) se ponavlja. Očekivani broj pokušaja bojenja je manji ili jednak od dva. Zaista, verovatnoća da se ispravno bojenje pronađe u j-tom pokušaju je $(1-p)^{j-1}p$, pa je matematičko očekivanje broja pokušaja jednako:

$$\begin{aligned}
\sum_{j=1}^{\infty} j \cdot (1-p)^{j-1}p &= \sum_{j=0}^{\infty} j \cdot (1-p)^{j-1}p = p\sum_{j=0}^{\infty} j(1-p)^{j-1} \\
&= -p\sum_{j=0}^{\infty} \Big((1-p)^j\Big)'_p = -p\Big(\sum_{j=0}^{\infty}(1-p)^j\Big)'_p \\
&= -p\Big(\frac{1}{p}\Big)'_p = -p\Big(-\frac{1}{p^2}\Big) = \frac{1}{p} \leq 2
\end{aligned}$$

U ovom nizu jednakosti i nejednakosti koristimo činjenicu da je $1 - p < 1$, kao i da je $p \geq \frac{1}{2}$.

Opisani algoritam bojenja je očigledno Las Vegas algoritam, jer se bojenja proveravaju jedno za drugim, a sa traženjem se završava kad se naiđe na ispravno bojenje. Ne postoji garancija uspešnog bojenja u bilo kom fiksiranom broju pokušaja, ali je ovaj algoritam u praksi ipak vrlo efikasan.

Provera matričnog množenja

Pretpostavimo da nam je na raspolaganju modul za množenje matrica i da želimo da proverimo da li on radi korektno za neki konkretan ulaz, tj. da za matrice A, B i C dimenzije $n \times n$ proverimo da li važi $AB = C$. Deterministički algoritam bi nad matricama A i B pokrenuo neki od standardnih algoritama za množenje matrica i nakon toga poredio svaki element matrice AB sa odgovarajućim elementom matrice C. Podsetimo se da je direktni algoritam za množenje matrica složenosti $O(n^3)$, dok je Štrasenov algoritam nešto bolje složenosti $O(n^{\log_2 7})$. Pokazuje se da je korišćenjem tehnike randomizacije moguće doći do odgovora u vremenskoj složenosti $O(n^2)$ sa jako malom verovatnoćom pogrešnog odgovora.

Ideja je izabrati na slučajan način izabrati n-dimenzioni vektor, pomnožiti njim obe strane ove matrične jednačine i videti da li je dobijeni rezultat isti. Preciznije, biramo binarni vektor $r = (r_0, r_1, \ldots, r_{n-1}) \in \{0, 1\}^n$ na slučajan način. Ukoliko je $(AB)r \neq Cr$, onda kao odgovor vraćamo da su matrice AB i C različite, a inače vraćamo da su jednake.

Ovaj algoritam je vremenske složenosti $O(n^2)$, jer je množenje matrica asocijativno, pa se računanje $(AB)r$ može izraziti kao $A(Br)$ čime se algoritam svodi na samo tri množenja matrice vektorom, što je ukupne vremenske složenosti $O(n^2)$. Jasno je da ako se dobije $(AB)r \neq Cr$, onda je odgovor sigurno ne. Međutim, ako su dobijeni vektori jednaki, može se desiti da nismo imali sreće sa izborom vektora r i da je ispravan odgovor da matrice nisu jednake. Međutim u nastavku ćemo pokazati da ako je $AB \neq C$ onda važi da je verovatnoća da se za slučajno odabran vektor r dobije da su odgovarajući proizvodi isti ograničen:

$$P[A \cdot B \cdot r = C \cdot r] \leq \frac{1}{2}$$

Neka je $D = C - AB \neq 0$. Potrebno je pokazati da je $P[Dr \neq 0] \geq \frac{1}{2}$. S obzirom na to da je $D \neq 0$, postoji neko $0 \leq j < n$ tako da D ima nenula vrednost u j-toj koloni. Fiksirajmo to j. Za proizvoljni vektor r, definišemo vektor r' kao vektor dobijen od r izmenom j-tog bita – ako je bio 0 postavljamo ga na 1, a ako je bio 1 na 0. Dakle važi $r' = r + e_j$ ili $r' = r - e_j$, gde je e_j j-ti vektor standardne baze koji ima 1 na mestu j-te koordinate, a sve ostale nule. Primetimo da važi $(r')' = r$. Ako definišemo graf G čiji čvorovi odgovaraju n-torkama $\{0, 1\}^n$ i postavimo granu između r i r' za svako r, onda skup grana grafa G predstavlja savršeno uparivanje u grafu. Tvrdimo da ako su r i r' upareni

u G, onda ne može istovremeno da važi $Dr = 0$ i $Dr' = 0$, tj. mora da važi bar jedno od $Dr \neq 0$ i $Dr' \neq 0$. Ovo važi zato što ako je $Dr = 0$, onda je $Dr' = D(r \pm e_j) = Dr \pm De_j = \pm De_j$. Međutim, De_j je baš j-ta kolona matrice D koja je po pretpostavci različita od 0. Dakle, za proizvoljni vektor r važi bar jedno od $Dr \neq 0$ i $Dr' \neq 0$, te za bar polovinu vektora r važi $Dz \neq 0$ i odavde sledi prethodno tvrđenje.

Da bi se smanjila verovatnoća greške, moguće je algoritam ponoviti k puta za k slučajno izabranih binarnih n-torki. Na taj način se verovatnoća da dođe do greške smanjuje na $(\frac{1}{2})^k$.